DICK SMITH VZ200

Personal Colour Computer

Further Programming

By

Tim Hartnell

B-7208

Further Programming

with your

Dick Smith VZ200 Personal Colour Computer

[illegible]

Tim Hartnell

FURTHER PROGRAMMING WITH YOUR DICK SMITH VZ200

By

Tim Hartnell

First published in Australia by:

Dick Smith Management Pty. Ltd.,
PO Box 321,
North Ryde, NSW, 2113

First printing: September, 1983

ISBN 0 949772 23 2

Dick Smith catalogue no. B-7210

Typesetting by APL Printers, Huntingdale
Artwork by Interface Publications, Level 25, Nauru House, 80 Collins St., Melbourne, 3000

CONTENTS

Poke 30744,0 white (or) Poke 30744,1 black

FOREWORD

Now that you've mastered simple BASIC on your computer, what do you do next?

The answer lies in this book. Tim Hartnell has covered the areas of computer programming which I think are most likely to be of interest to you, including moving graphics, the use of MODE(1), music and PEEK and POKE.

As well as the 'lessons', there are fifteen or so games which you're sure to enjoy running. The appendix contains a useful glossary of computer words, and a very clear explanation of how computers in general - and yours in particular - work.

Now that you have simple programming under control, it is time to tackle some more difficult, but highly rewarding, areas. This book gives you the key.

Jeremy Ruston,
1983

*Jeremy Ruston is the author of a number of popular computer books, including PASCAL FOR HUMAN BEINGS (Interface Publications, 1982), THE BOOK OF LISTINGS (co-author Tim Hartnell, BBC Publications), THE BBC MICRO REVEALED (Interface Publications, 1982) and THE BBC MICRO COMPENDIUM (Interface Publications, 1982). He contributes extensively to the computer press, including POPULAR COMPUTING WEEKLY, PERSONAL COMPUTING TODAY and ZX COMPUTING.

CHAPTER ONE — CREATING PROGRAMS

Many books on programming suggest you start by drawing up a 'flowchart' - a combination of circles, diamonds and slanting rectangles, which sets out the path and operations the computer will follow to execute a program. In theory, this is fine, but in many cases the time and trouble involved may not be worth it.

It is, however, essential that you know exactly what you want the computer to do before you start creating a new program, even if you're not quite sure how you're going to get the computer to carry out the task.

Sometimes a rough sort of flowchart - perhaps just a list of the main steps the computer will take, linked by little arrows and loops - will help to clarify your thinking. It is also a good way to spot potential problems, like the danger of setting up infinite loops, or of not specifying the nature of the computer's decisions exactly.

I would like to introduce you to a school of programming thought called 'Structured Programming'. Structured programming says that programs need to be designed from the 'top down'. That is, you need to spell out in words, long before you start actual programming, exactly what you want the computer to do, and in what order.

The first program in the book, TIC TAC TOE, was written to demonstrate structured programming in action. If you look at the beginning of the listing, lines 130 to 170, you'll see what is - in effect - the whole program in a series of subroutine calls.

```
10 REM TIC TAC TOE
20 CLS
90 DIM A(9)
95 FOR T=1 TO 9:A(T)=0:SOUND T,1:NEXT T
100 T=RND(0)
105 IF T>.7 THEN PRINT "I'LL HAVE THE FIRST MOVE"
110 IF T<=.7 THEN PRINT "YOU HAVE THE FIRST MOVE"
115 FOR Z=1 TO 1000:NEXT:CLS
120 IF T>.7 THEN 160
125 IF INKEY$<>"" THEN 125
130 GOSUB 750
140 GOSUB 480
150 GOSUB 670
160 GOSUB 750
170 GOSUB 480
180 IF A(5)=0 THEN A(5)=1:GOTO 130
```

What do I mean by this claim, the 'whole program'? When I sat down to write this program, I first listed the steps the computer would follow, like this:

- decide who goes first
- print out the board
- check to see if anyone has won
- let the human move
- print out the board
- check to see if anyone has won
- let the computer move
- go to the second item in the list and cycle through again and again until somebody wins

Once I had this list written, I changed most of the instructions into subroutine calls. Look again at the lines I mentioned. By following them through in the program, you'll see they correspond fairly closely to my initial list:

10 - 115: Decide who will have first move

120: If the computer is having first move, jump to line 160, which calls up the board print subroutine

130: If not, this line goes to the subroutine from line 750 which prints up the board

140: Sends action to the subroutine from line 480 which performs the 'win check'

150: Calls the subroutine from line 670 which accepts the player's move

160: Calls the board print subroutine

170: Calls the win check subroutine

180: Moves into the centre square if it's available, then goes back to 130 to cycle through again

You'll see, in the next section, that a number of lines end with GOTO 130. These are triggered when the computer has moved, and the program then goes back to 130 to cycle again. Apart from the 'who will go first?' routine which was added later, the program proper is really held within those few lines 130 to 170. These were written as subroutine calls, and each subroutine labelled, long before I had actually written the relevant subroutines.

This meant that I could proceed step by step, debugging an isolated subroutine, before going on to the next. This is the real advantage of structured programming. Because the program consists of a number of discrete modules, each charged with performing a specific task, tracking down bugs is simplified, the flow of action through the program is relatively transparent, and subsequent modification of the program is easy.

```
190 REM TO COMPLETE ROW/BLOCK
200 D=1
210 B=1
220 IF B=1 THEN X=1:Y=2:Z=3
230 IF B=2 THEN X=1:Y=4:Z=7
240 IF B=3 THEN X=1:Y=5:Z=9
250 IF B=4 THEN X=3:Z=7
260 C=1
270 IF A(X)=D AND A(Y)=D AND A(Z)=0 THEN A(Z)=1:GOTO 130
280 IF A(X)=D AND A(Y)=0 AND A(Z)=D THEN A(Y)=1:GOTO 130
290 IF A(X)=0 AND A(Y)=D AND A(Z)=D THEN A(X)=1:GOTO 130
300 IF B=1 THEN X=X+3:Y=Y+3:Z=Z+3
310 IF B=2 THEN X=X+1:Y=Y+1:Z=Z+1
320 IF C<3 THEN C=C+1:GOTO 270
330 IF B<4 THEN B=B+1:GOTO 230
340 IF D<2 THEN D=D+1:GOTO 210
350 REM MOVE AT RANDOM
360 B=1
370 D=INT(RND(0)*9)+1
380 IF A(C)=0 THEN A(C)=1:GOTO 130
390 B=B+1
400 IF B<21 THEN 370
410 B=0
```

```
420 B=B+1
430 IF A(B)=0 THEN A(B)=1:GOTO 130
440 IF B<9 THEN 420
450 GOSUB 750
460 PRINT:PRINT"IT'S A DRAW"
470 GOTO 650

480 REM WIN CHECK
490 FOR B=1 TO 4
500 IF B=1 THEN X=1:Y=2:Z=3
510 IF B=2 THEN X=1:Y=4:Z=7
520 IF B=3 THEN X=1:Y=5:Z=9
530 IF B=4 THEN X=3:Z=7
540 FOR C=1 TO 3
550 IF A(X)=A(Y) AND A(Y)=A(Z) AND A(X)<>0 THEN 610
560 IF B=1 THEN X=X+3:Y=Y+3:Z=Z+3
570 IF B=2 THEN X=X+1:Y=Y+1:Z=Z+1
580 NEXT C
590 NEXT B
600 RETURN

610 REM THE WINNER!
620 PRINT
630 IF A(X)=1 THEN PRINT "I'M THE WINNER!"
640 IF A(X)=2 THEN PRINT "YOU'RE THE WINNER!"
650 FOR D=1 TO 1000:NEXT D
660 GOTO 95
670 REM PLAYER MOVE
675 IF INKEY$<>"" THEN 675
680 PRINT @ 450,"ENTER YOUR MOVE"
690 A$=INKEY$
700 IF A$<"1" OR A$>"9" THEN 690
710 B=VAL(A$)
720 IF A(B)<>0 THEN 690
725 PRINT @ 450,"                "
730 A(B)=2
740 RETURN
```

```
750 REM PRINTOUT
760 PRINT @ 64,
780 PRINT "1 2 3   ";
790 FLA =0
800 FOR B=1 TO 9
810 IF A(B)=0 THEN FLA =1
820 IF A(B)=0 THEN PRINT " - ";
830 IF A(B)=1 THEN PRINT " O ";:SOUND 1,1
840 IF A(B)=2 THEN PRINT " X ";:SOUND 15,1
850 IF B=3 THEN PRINT:PRINT:PRINT "4 5 6   ";
860 IF B=6 THEN PRINT:PRINT:PRINT "7 8 9   ";
870 NEXT B
880 IF FLA =0 THEN 460
890 RETURN
```

Here's the program in action:

```
1 2 3     -  -  -

4 5 6     -  -  -

7 8 9     -  -  -


1 2 3     -  -  -

4 5 6     -  X  -

7 8 9     -  -  -


1 2 3     -  -  O

4 5 6     -  X  -

7 8 9     -  -  -


1 2 3     -  -  O

4 5 6     -  X  -

7 8 9     -  -  X
```

```
1 2 3     O  -  O

4 5 6     -  X  -

7 8 9     -  -  X


1 2 3     O  X  O

4 5 6     -  X  -

7 8 9     -  -  X


1 2 3     O  X  O

4 5 6     -  X  -

7 8 9     -  O  X


1 2 3     O  X  O

4 5 6     -  X  -

7 8 9     X  O  X


1 2 3     O  X  O

4 5 6     O  X  -

7 8 9     X  O  X


1 2 3     O  X  O

4 5 6     O  X  X

7 8 9     X  O  X


      IT'S A DRAW!
```

```
1 2 3    - - -
4 5 6    - O -
7 8 9    - - -
                 1 2 3    - - -
                 4 5 6    - O -
                 7 8 9    - X -
1 2 3    - - O
4 5 6    - O -
7 8 9    - X -
                 1 2 3    - - O
                 4 5 6    - O -
                 7 8 9    X X -
1 2 3    - - O
4 5 6    - O -
7 8 9    X X O
                 1 2 3    X - O
                 4 5 6    - O -
                 7 8 9    X X O
```

```
1 2 3    X - O
4 5 6    - O O
7 8 9    X X O

I'M THE WINNER!
```

I urge you to follow a sequence of steps such as those outlined above when you're writing a program from scratch. It may not be necessary to do so if you're working on a fairly simple program, or are adapting from a magazine or book. Even then, if you're working directly on the computer, it is handy to keep a notebook by you to record such things as that you've assigned A$ to the player's name.

GETTING THINGS ON TAPE

If you're writing a program directly on the keyboard, and you want to add a subroutine which will eventually be at the end of program, give it a very high line number. It can easily be renumbered later if you like. This is far better than having to jump over a subroutine, as the program expands, with an ugly GOTO.

You should resist the temptation to put all your programs on a single cassette, one program after another. The frustration you'll experience searching through the tape - even if you've identified each program with a voice label - to find a particular program, is just not worth the trouble.

Go to your computer store, and get a set of C-12 or C-20 cassettes. Put just one program on each cassette, with two or three copies of each one in case your recording gets damaged, accidentally erased or won't load.

Write the name of the program on the cassette and on the cardboard insert. You'll find this makes it very easy to locate the program you want. As well, as your library of programs grows, it will give you quite a feeling of accomplishment to see all those programs ranked side by side. It will impress your family and friends as well, who will believe after visiting you and playing with your computer that you're some kind of natural computer genius (which you probably are).

CHAPTER TWO — MOVING GRAPHICS

There is little doubt that moving graphics games, such as the ones you see in arcades, are among the most exciting things you can do with your computer. Although the programs we create in this section of the book will be in no way as sophisticated as arcade games, the techniques given will allow you to write moving graphics games of your own, and will give you an insight into arcade-standard programs.

THE SECRET OF ANIMATION

It is simple to make something appear to move on the computer screen. You put a shape - such as the letter A - on the screen in a particular spot, and hold it there for a moment or two. Then, you print a blank space where the A was, and at the same time, reprint the A a short distance away from its original position.

This process is repeated over and over again, and it creates the impression that the object is moving. That's all there is to it. Put one moving object under your control, and another one under the computer's control, and you have the raw ingredients of a game. Let's see how it works in practice.

USING PRINT @

As you know, the computer's screen is 32 characters across, and 16 lines down. The PRINT @ command works by counting the very first position on the screen (the one in the top, left hand corner) as number 0. It counts across the line, and then gets to position 31 at the end of the line. Position 32 is the first one on the second line. This goes right through the screen with the final position (in the bottom, right hand corner) as position number 511.

We'll start by placing something on the screen. Enter this brief program and run it. When you see the question mark appear, enter two numbers separated by a comma. The first is the number of lines down you want the object to appear, and the second is the number of spaces you want the object to be across the screen. The first number must be between 0 and 15, and the second number between 0 and 31.

Enter the program, run it for a while, then return to the book:

```
10 REM PRINT @ DEMO
20 CLS
30 INPUT A,B
40 PRINT @ (32*A+B),"A"
50 PRINT @ 0,;
60 GOTO 30
```

Note the use of the formula in line 40, which changes the numbers entered as down and across co-ordinates into a single number which the PRINT @ command can use (later on we'll use a similar formula for PEEK and POKE, but we'll stay with PRINT @ for the time being):

PRINT @ (32*A + B)

You'll find you're using this formula, or a

similar one to it, time and time again in programs.

We'll now use PRINT @, and our formula, to create a bouncing ball program. NEW the computer, and enter the following program, then return to the book for a discussion on it. You'll find that the techniques used for animating this ball will be used in just about every moving graphics game you write. So, it is worth taking the trouble to understand exactly how it works, so you can apply the techniques to your own programs.

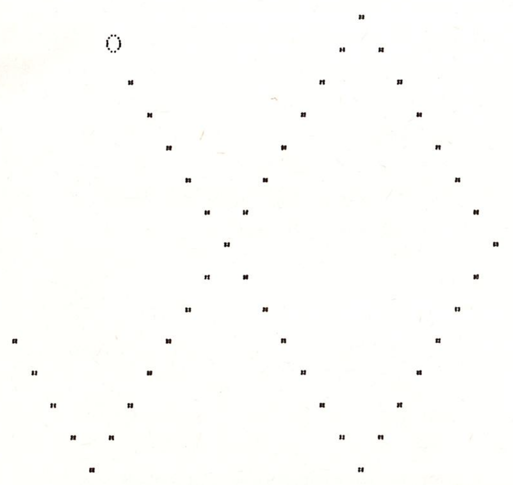
```
10 REM BOUNCING BALL 1
15 CLS
20 A=6:B=11
30 Y=1:X=1
40 EA=A:EB=B
60 PRINT @ (32*B + A),"O"
70 B=B+X
80 A=A+Y
90 IF A<2 THEN Y=-Y
100 IF A>30 THEN Y=-Y
110 IF B<2 THEN X=-X
120 IF B>14 THEN X=-X
130 GOTO 40
```

When you run this, you'll see the ball appear on the screen, bouncing from the sides, and leaving a trail of itself behind. I deliberately did not erase the 'old' ball in this program, so you could see that it was just a series of the same character being printed on the screen.

Now, add this line, and see what happens:

```
125 PRINT @ (32*EB + EA),"."
```

When you run the program, you'll see something like this on the screen:



As you can see, it leaves a trail of dots showing where the ball has been. Remove the dot from between the quote marks in line 125, and run the program again, to see a true bouncing ball.

The program gives a very good impression of a moving object. This program, as I said earlier, demonstrates a number of key things about moving graphics programs, so we'll go through this program line by line, explaining what each line is doing.

15 - clears the screen

20 - sets the start position of the ball, at 7 across and 12 down

30 - sets X and Y which control the change in position of the ball between each movement

40 - sets two variables used to erase the old position. These are called EA (for 'erase A') and EB ('erase B). You should always use variable names which suggest the function the variable is performing, as it makes it much simpler later on when you're going through a program. Note (and this is important) EA and EB are set equal to A and B just before the ball is printed at 32*B + A (line 60) and before A and B are modified (lines 70 and 80)

60 - this prints the ball on the screen

70 - B (the down position) is changed

80 - A (the across position) is changed

90 - this checks that the A position is not too far to the left, and if it is, changes Y to minus Y (causes the ball, next time it comes to this line, to bounce off the left hand side of the screen)

100 - does the same for the right hand side

110 - checks the up/down position, and if the ball is too close to the top of the screen, changes X to minus X which, as you'd guess,

causes the ball to bounce down from the top of the screen

120 - does the same for the bottom of the screen

125 - prints a blank on the 'old' position of the ball (that is, the position designated by A and B before they were modified by lines 70 and 80)

130 - sends action back to line 40, where EA and EB are set equal to the ball's new position, before it is reprinted in the new position with line 60

Although this explanation is fairly long (and much longer than the program it is explaining) it is worth reading it through carefully. Once you understand it, you'll be well on the way to writing your own programs.

Now, modify the program so it is as follows. In this version, there is a little object at the bottom of the screen which is your 'bat' and you have to get the ball to bounce off it to keep the ball moving. You use the 'Z' (to move left) and the 'M' (to move right) keys to control the bat at the bottom.

```
10 REM BOUNCING BALL 11
15 CLS
20 A=6:B=11:F=11
30 Y=1:X=1
40 EA=A:EB=B
50 PRINT @ (477+F),"  ";CHR$(131);"  "
```

```
60 PRINT @ (32*B + A),"O"
70 B=B+X
80 A=A+Y
82 A$=INKEY$
85 IF A$="Z" AND F>3 THEN F=F-2
86 IF A$="M" AND F<29 THEN F=F+2
90 IF A<2 THEN Y=-Y
100 IF A>30 THEN Y=-Y
110 IF B<2 THEN X=-X
120 IF B>13 AND ABS(F-A)<3 THEN X=-X
121 IF B=16 THEN END
125 PRINT @ (32*EB + EA)," "
130 GOTO 40
```

Now, this is not a particularly satisfying game (it probably doesn't even deserve the title 'game') but it shows some extra ingredients you'll be using in moving graphics games. In this program outside interaction is brought into play for the first time (you moving the bat) and the computer responds to this (keeping the game underway if the bat is in the right position).

Here's what the extra lines do:

20 - F is the position of the bat across the screen

50 - this prints the 'bat' (and you can use the graphics character direct from the Y key instead of CHR$(131) and note there must be two spaces within the quote marks on either side, to 'unprint' the bat as it moves)

82 - this reads the keyboard, with INKEY$, and

sets the result of this reading equal to A$

85 - if A$ equals "Z" (that is, if you are pressing the "Z" key) and F is greater than 3, the value of F is reduced by 2

86 - if A$ equals "M" and F is less than 29, then the value of F is increased by 2

(The effect of these two lines, as I'm sure you can see, is to move the bat in accord with your wishes.)

120 - if B is greater than 13 (which means the ball is near the bottom of the screen) the position of the bat is checked, and if it is close to the ball, a bounce occurs and the game continues

121 - if the ball has missed the bat, this line halts the program

This is not, as we said, a very inspiring or challenging game, but with a few minor changes can be given a bit of value. Modify it so it reads as follows, and play with it a bit:

```
10 REM BOUNCING BALL 111
12 REM 'SQUASH'
15 CLS
17 PRINT @ 224,"--------------------------------"
18 FOR J = 256 TO 448 STEP 32
19 PRINT @ J,"     >                    <":NEXT
20 A=6:B=11:F=11
25 SC=1
30 Y=1:X=1
```

```
40 EA=A:EB=B
50 PRINT @ (477+F),"  ";CHR$(131);"  "
60 PRINT @ (32*B + A),"*"
70 B=B+X
80 A=A+Y
82 A$=INKEY$
85 IF A$="Z" AND F>8 THEN F=F-2
86 IF A$="M" AND F<24 THEN F=F+2
90 IF A<7 THEN Y=-Y
100 IF A>23 THEN Y=-Y
110 IF B<9 THEN X=-X
120 IF B=15 AND ABS(F-A)<2 THEN X=-X
121 IF B=16 THEN END
122 IF B=15 THEN SC=SC+367:PRINT @ 68,"SCORE IS ";SC
125 PRINT @ (32*EB + EA)," "
130 GOTO 40
```

83+08

23122

2203

7341

Again you'll have to use the "Z" and "M" keys, but this time you have a much smaller 'playing field' and there is a bit of a challenge to the game. You might like to spend some time playing with this program, modifying it as you see fit, before returning to the book to discuss moving an object up and down, as well as to the right and left, on the screen.

TRAPPER

6950

We now have a program which is difficult (and very enjoyable) to play, as the 'X' is stalked by the prehistoric '@'.

In this program - TRAPPER - you are the "X" and the computer controls the "@". You use the following keys to move yourself around the screen, trying to stay out of the computer's clutches for as long as possible:

A - to move up
Z - to move down
M - to move left (see arrow above this key)
, - to move right (see arrow)

```
10 REM TRAPPER
15 CLS:T=0
20 BA=5:BD=5
30 TH=15:TV=15
40 EA=BA:ED=BD
50 EH=TH:EV=TV
60 PRINT @ (32*ED+EA)," "
70 PRINT @ (32*TV+TH)," "
80 A$=INKEY$
90 IF A$="A" AND BD>2 THEN BD=BD-1
100 IF A$="Z" AND BD<14 THEN BD=BD+1
110 IF A$="," AND BA<30 THEN BA=BA+1
120 IF A$="M" AND BA>2 THEN BA=BA-1
130 PRINT @ (32*BD+BA),"X"
135 IF RND(0)>.5 THEN 150
140 IF BD>TV THEN TV=TV+1
145 IF RND(0)>.5 THEN 160
150 IF BD<TV THEN TV=TV-1
155 IF RND(0)>.5 THEN 170
160 IF BA<TH THEN TH=TH-1
170 IF BA>TH THEN TH=TH+1
180 PRINT @ (32*TV+TH),"@"
185 T=T+1
190 PRINT @ 0,"TIME ELAPSED"T
210 IF BD=TV AND BA=TH THEN SOUND RND(0)*20,1:GOTO 210
220 GOTO 40
```

There is no need to go through this program line by line, as I'm sure by now you've got a pretty good idea of what effect the various parts of the program have. The main difference between TRAPPER and SQUASH is that we are now moving in two dimensions, and the thing which is trying to trap us can 'sense' (using lines 135 to 170) just where we are. The lines 135, 145 and 155 are just to give you a chance. Take them out, and you'll be dead within 10 moves, every single time.

This program points up one of the disadvantages of working in BASIC, especially with PRINT @ - it is fairly slow, and everything we add to a program slows it down further. Fortunately, there is another way to move things on the screen, and we'll turn to that now.

PEEK and POKE

Although PEEK and POKE seem to inspire fear into programmers when they first come across them, there is no need to worry. They can be used in almost exactly the same way as PRINT @ has been used.

There are, however, two advantages of using PEEK and POKE over using PRINT @:

- POKE is faster than PRINT @
- PEEK can be used to find out what is on the screen in a particular position. This is ideal for discovering whether or not an alien has been shot, or the plane you are flying has hit the side of a mountain

When you POKE something to the screen, you put it on the screen. When you PEEK, you are looking to see what is on the screen at the particular position. That's just about all you have to remember.

If you have two co-ordinates A (for across) and D, you PRINT @ them, as explained before, using the formula 32*D + A. To POKE something into place, you use a line like:

```
POKE 28672 + 32*D + A, X
```

We generally use 28672, the location of the top left hand corner of the screen, or 28736, the position two lines below it. The X is the character number, and this is the character which will appear when we POKE directly to the screen.

You'll remember that we used a space (" ") in PRINT @ to wipe something out. The equivalent in this case is the number 32. So...

```
POKE 28672 + 32*D + A, 32
```

...is the equivalent of ...

```
PRINT @ (32*D + A)," "
```

If you want to find our what is at a particular address, you use PEEK as follows:

```
IF PEEK (28672 + 32*D + A) = .... THEN ....
```

Now, PEEK and POKE are simpler to use than they

may appear at first sight. I suggest you enter the following programs, all of which use the two commands, and follow through the listings carefully. This will allow you to learn about the use of the two words more quickly than any other way.

MESOZOIC ATTACK

The Mesozoic Era was when dinosaurs roamed the earth. This program is a PEEK/POKE version of TRAPPED (although only POKE is used, to break you into the new words gradually). Note that although up and down are the same keys ("A" and "Z") right and left are "," (left) and "." (right). It is easy to remember these, as the greater than and less than symbols point in the relevant directions.

```
10 REM MESOZOIC ATTACK
15 HS=0
20 CLS
30 SC=0
40 HA=1:HD=1:D=15:A=30
50 EA=HA:ED=HD
60 IF INKEY$="A" THEN HA=HA-1
70 IF INKEY$="Z" THEN HA=HA+1
80 IF INKEY$="." THEN HD=HD+1
90 IF INKEY$="," THEN HD=HD-1
91 IF HA<1 THEN HA=1
92 IF HA>15 THEN HA=15
93 IF HD<1 THEN HD=1
94 IF HD>31 THEN HD=31
95 EB=SB
100 SB=28672 + 32*D+A
110 IF SB=H THEN GOTO 1000
115 POKE EB,32
```

```
120 POKE SB,(36+INT(RND(0)*3))
130 H=28672 + 32*HA+HD
140 IF SB=H THEN GOTO 1000
145 POKE EH,32
150 POKE H,88:EH=H
160 IF D<HA THEN D=D+1
165 IF RND(0)>.3 THEN 180
170 IF D>HA THEN D=D-1
175 IF RND(0)>.3 THEN 190
180 IF A<HD THEN A=A+1
190 IF A>HD THEN A=A-1
200 SC=SC+1
210 PRINT @ 20,"TIME>"SC
240 IF D<1 THEN D=1
250 IF D>15 THEN D=15
260 IF A<1 THEN A=1
270 IF A>31 THEN A=31
275 IF SB=H THEN GOTO 1000
280 GOTO 50
1000 POKE SB,32:POKE H,32
1001 POKE EB,32:POKE EH,32
1002 FOR J=1 TO 19
1003 POKE SB,(80+INT(RND(0)*20))
1005 PRINT @ 485, "I GOT YOU!!!!!!"
1010 SOUND RND(20)+1,1
1020 NEXT J
1030 PRINT @ 226,"YOU SURVIVED FOR";SC
1035 IF SC>HS THEN HS=SC
1040 PRINT @ 258,"BEST SO FAR IS";HS
1050 FOR T= 1 TO 500:NEXT T
1060 GOTO 20
```

V-WING SPACE BATTLE

In this program, you pilot your little V-wing space craft around the screen, trying to run over the numbers which appear on it. Each number will appear for a limited amount of time, and if you run over it, a score related to that number is added to your growing tally.

PEEK is used here for the first time, in line 190, where it looks at the position where your space craft is about to be POKEd, and if it finds a number there, increments your score.

```
10 REM V-WING SPACE BATTLE
20 CLS
30 SC=0
100 FOR Z= 1 TO 20
110 SL=28736:M=22:D=-32
120 POKE 28715+INT(RND(0)*468),INT(RND(0)*9)+49
130 W=SL
140 A$=INKEY$
150 IF A$="," THEN SL=SL+1:M=62
160 IF A$="M" THEN SL=SL-1:M=60
170 IF A$="." AND SL>28736 THEN SL=SL-32:M=1
180 IF A$=" " AND SL<29151 THEN SL=SL+32:M=22
190 Q=PEEK(SL)
200 IF Q>48 AND Q<58 THEN SC=SC+Q:GOTO 980
205 POKE W,32
210 POKE SL,M
220 IF RND(0)<.99 THEN 130
230 COLOR,1:FOR T=1TO20:NEXT T:COLOR,0
980 CLS:PRINT @ 0,"SCORE  ";SC;"   ";20-Z;" SHIPS LEFT"
982 COLOR,INT(RND(0)*2)
985 SOUND RND(0)*25+1,1:IF RND(0)>.6 THEN 985
990 NEXT Z
```

```
1000 PRINT @ 0,"THE BATTLE IS OVER","YOU SCORED ";SC
1100 COLOR,INT(RND(0)*2)
1120 GOTO 1100
```

BREAKOUT

As you can see, the programs are getting more involved. BREAKOUT produces a screen picture like this:

```
 ################################

   X  X  X  X  X  X  X  X  X  X  X  X  X  X  X
 X  .  X  .  X  .  X  .  X  .  X  .  X  .  X  .
   X  .  X  .  X  .  X  .  X  .  X  .  X  .  X
      X  X  X                X                X



 THE GAME IS OVER

 YOU SCORED  5244



 SCORE  5244            BALLS  0
```

You have to knock as many bricks (the X's) out of the way as you can, trying to get through to the row of hash symbols near the top of the screen. PEEK is used (see lines 140 through to 146) to check whether you've hit a brick (140, 144 and 146) or the hash symbols (145). Your bat at the bottom of the screen is POKEd into position by lines 225 and 230 (with the two POKE 32's clearing each side of the bat).

```
10 REM BREAKOUT
15 CLS
20 PRINT"################################"
25 PRINT " X X X X X X X X X X X X X X X "
30 PRINT "X . X . X . X . X . X . X . X ."
40 PRINT " X . X . X . X . X . X . X . X "
50 PRINT "X X X X X X X X X X X X X X X X "
60 CX=10:REM NO. OF BALLS
70 BA=INT(RND(0)*6)+6:REM START OF BALL ACROSS
80 BD=8: REM START OF BALL DOWN
82 F=5 +INT (RND(0)*10):REM START OF BAT ACROSS
85 X=1.1:Y=1
90 EA=BA:ED=BD:REM ERASE BALL
100 POKE 28672+(32*ED+EA),32
110 POKE 28672+(32*BD+BA),48:REM PLACE BALL
115 EA=BA:ED=BD:REM ERASE BALL
120 IF BA>30 OR BA<1 THEN GOSUB 1000
130 IF BD>11 THEN GOTO 2000
140 IF PEEK (28640+32*BD+BA)=24 THEN GOSUB 3000:GOTO 150
144 IF PEEK (28640+32*BD+BA+1)=24 THEN GOSUB 3000:GOTO 150
145 IF PEEK (28640+32*BD+BA)=35 THEN GOTO 4000
146 IF PEEK (28640+32*BD+BA-1)=24 THEN GOSUB 3000
150 BA=BA+X
160 BD=BD+Y
200 A$=INKEY$
210 IF A$="Z" THEN F=F-1
220 IF A$="M" THEN F=F+1
222 IF F<1 THEN F=1
223 IF F>30 THEN F=30
225 S=29088+F
230 POKE S-1,32:POKE S+1,32:POKE S,140
500 GOTO 100
1000 X=-X
1200 RETURN
2000 IF ABS(F-BA)>2 THEN 2500
2005 Y=-Y
2200 GOTO 140
```

```
2500 CX=CX-1
2505 IF CX=0 THEN PRINT @ 288,"THE GAME IS OVER"
2506 IF CX=0 THEN PRINT @ 352,"YOU SCORED ";SC
2510 PRINT @ 480,"SCORE ";SC;"       BALLS ";CX
2520 PRINT @ 416,"                                "
2530 PRINT @ 384,"                                "
2535 IF CX=0 THEN 4020
2537 FOR T=1 TO 10:SOUND T,1:NEXT
2540 BA=INT(RND(0)*6)+6:REM START OF BALL ACROSS
2550 BD=8: REM START OF BALL DOWN
2560 F=5 +INT (RND(0)*10):REM START OF BAT ACROSS
2570 GOTO 90
3000 POKE (28640+32*BD+BA),32
3001 IF RND(0)>.8 THEN POKE (28640+32*BD+BA+1),32
3002 IF RND(0)<.1 THEN POKE (28640+32*BD+BA-1),32
3010 SC=SC+437
3020 PRINT @ 480,"SCORE ";SC
3030 COLOR,1:FOR T=1TO5:NEXT:COLOR,0
3035 Y=-Y
3040 RETURN
4000 PRINT @ 288,"YOU HAVE DONE IT!"
4010 PRINT @ 352,"YOU SCORED ";SC*CX
4020 REM END OF GAME
4025 FOR T = 10 TO 30
4030 SOUND T,1
4040 NEXT
4050 RUN
```

PLAGUE SPOT

In this gripping game, you have to work yourself (a graphics symbol, shown as the $ in the sample printouts) from the top left hand corner of the screen to the bottom right hand one.

The plague spots (the X's) are constantly

increasing as the game goes on, and each time you hit one your score will increase. You have to get to the safe spot in the bottom right hand corner with the lowest possible score.

Here are a few 'snapshots' of it in action:

```
 950    XXXXXXXXXXXXXXXXXXXXXXXX
X                  X         X X
 XXX              X  X      X  X
X   XX X     X            X  X X
X           X    X X           X
X X                       X    X
X      X           X           X
X X                            X
X   X    X X            XX     X
X                     X        X
XX           X       $X X   X  X
X X        X           X X  XX X
X     X      X   X   X         X
X        X                  XX X
X                              X
XX         960    XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXX X                  X         X X
           X X              X  X      X  X
          X   XX X     X            X  X X
          X           X    X X           X
          X X                            X
          X      X           X           X
          X X                     XX     X
          X   X      X   $      X        X
          XX           X X      X     X  X
          X X        X           X X  XX X
          X            X   X   X         X
          X        X                  XX X
          X                              X
          XX          X   XX XX      XX XX
          XXXXXXXXXXXXXXXXXXXXXXXXXXXXX$$
```

There is a high score feature, so you can try and get a better (that is lower) score each round. You'll find that adding a high score touch like this one will enhance many games.

```
10 REM PLAGUE SPOT
15 HS=9999999
20 CLS
30 SC=0
35 B$="."
40 PRINT @ 0,"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
50 FOR J=31 TO 447 STEP 32
60 PRINT "X                              X";
70 NEXT J
80 PRINT @ 480,"XXXXXXXXXXXXXXXXXXXXXXXXXXXXX$$$";
110 SL=28736
120 FOR T=1 TO 30
130 Z=28715+INT(RND(0)*468)
140 POKE Z,24
150 NEXT T
160 W=SL
170 A$=INKEY$
175 IF A$<>"A" AND A$<>"Z" AND A$<>"," AND A$<>"." THEN A$=B$
176 B$=A$
180 IF A$="." THEN SL=SL+1:IF PEEK (SL)=24 THEN GOSUB 1000
190 IF A$="," THEN SL=SL-1:IF PEEK (SL)=24 THEN GOSUB 1100
200 IF A$="Z" THEN SL=SL+32:IF PEEK (SL)=24 THEN GOSUB 1200
210 IF A$="A" THEN SL=SL-32:IF PEEK (SL)=24 THEN GOSUB 1300
215 IF SL <28672 THEN SL=28672
216 IF SL >29183 THEN SL=28672
219 IF PEEK(SL)=36 THEN 5000
220 POKE W,32:POKE SL,137
230 POKE 28715+INT(RND(0)*468),24
500 GOTO 160
1000 SOUND 1,1
1010 SL=SL-1
1020 SC=SC+1
```

1000 x 5 times
Perfect

```
1030 GOTO 1500
1100 SOUND 4,1
1110 SL=SL+1
1120 SC=SC+1
1130 GOTO 1500
1200 SOUND 8,1
1210 SL=SL-32
1220 SC=SC+1
1300 SOUND 12,1
1310 SL=SL+32
1320 SC=SC+1
1500 PRINT @ 0, 1000-10*SC;"   "
1505 IF SC>99 THEN 5000
1510 RETURN
5000 FOR T=1 TO 5
5005 PRINT @ 476,"###"
5010 PRINT @ 508,"^^^"
5020 SOUND RND(19)+1,1
5030 PRINT @ 476,"^^^"
5040 PRINT @ 508,"###"
5060 NEXT T
5070 IF SC<HS THEN HS=SC
5090 PRINT @ 192,"    YOUR SCORE WAS ";1000-10*SC
5100 PRINT @ 320,"    BEST SO FAR IS ";1000-10*HS
5120 FOR T=1 TO 15:SOUND T,1:NEXT
5130 GOTO 20
```

Finally, here are the character codes to use for PEEK and POKE:

0	@	16	P	32		48	0
1	A	17	Q	33	!	49	1
2	B	18	R	34	"	50	2
3	C	19	S	35	#	51	3
4	D	20	T	36	$	52	4
5	E	21	U	37	%	53	5

6	F	22	V	38	&	54	6
7	G	23	W	39	'	55	7
8	H	24	X	40	(	56	8
9	I	25	Y	41	)	57	9
10	J	26	Z	42	*	58	:
11	K	27	[	43	+	59	;
12	L	28	\	44	,	60	<
13	M	29	]	45	−	61	=
14	N	30	↑	46	.	62	>
15	O	31	←	47	/	63	?

CHAPTER THREE — USING MODE 1

As you know, your computer is equipped with two graphics modes, MODE (0) (which it is in automatically when you first turn the computer on) and the high resolution mode, MODE (1). We've been using MODE (0) to date, so it's about time we had a look at the other one.

SPIROGRAPH PATTERNS

Spirograph patterns are formed by both the interior and exterior epicycloid curves. One of the formulae for such a curve is given in line 100 of the first program in this section, in which A is the radius of the large circle, B is the radius of the small circle, and H is the point on the circumference of the small circle.

The program here chooses A, B and C at random, then traces out the shape on the screen. The finished pattern is held for a moment, then the process begins again.

If you don't like a particular pattern which is forming (or you can't see one at all, which happens with some combinations), just press any key and a new design will begin. Some of the results of this program are very, very attractive indeed. You'll find the patterns look best if viewed from a slight distance. This program is based on one written by Keith Hewson.

```
10 REM SPIROGRAPH
15 PI=3.141592
20 MODE(1)
25 COLOR INT(RND(0)*3)+2
```

```
30 IF INKEY$<>"" THEN 30
40 A=INT(RND(0)*15)+1
50 B=INT(RND(0)*15)+1
60 H=INT(RND(0)*15)+1
70 FOR J=0 TO 3*PI/1.7
80 FOR I=0 TO 2*PI STEP PI/60
90 X=(A-B)*COS(I)+H*COS((A-B)*I/B+J)
95 IF INKEY$<>"" THEN RUN
100 Y=(A-B)*SIN(I)+H*SIN((A-B)*I/B+J)
110 SET ((60+X),(32+Y))
120 NEXT I
130 NEXT J
140 FOR T=1 TO 1000
150 NEXT T
160 RUN
```

LISSAJOUS FIGURES

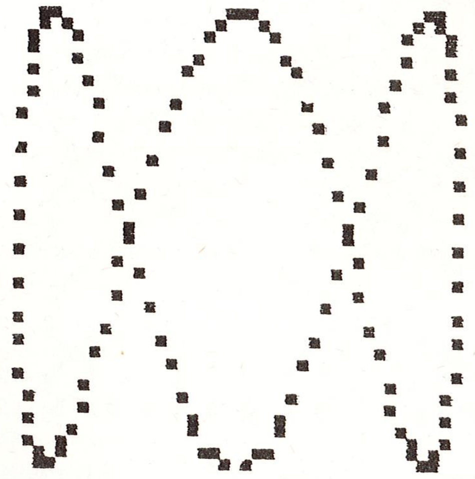
Jules Antoin Lissajous, a French physicist who lived from 1822 to 1880, made a study of the movement of particles under the action of periodic motions acting at right angles to each other. He discovered that bodies moving in this way trace intricate patterns as they dance around each other. This program, based on one written by Frazer Melton, shows what Lissajous discovered.

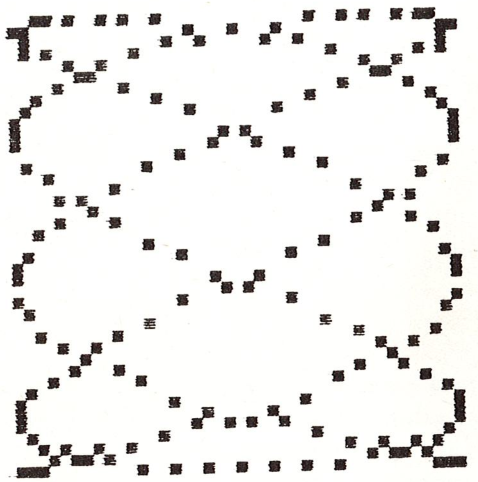
As in the SPIROGRAPH program we've just looked at, SET is used to plot the points on the screen, tracing out the path of the sum of the periodic motions.

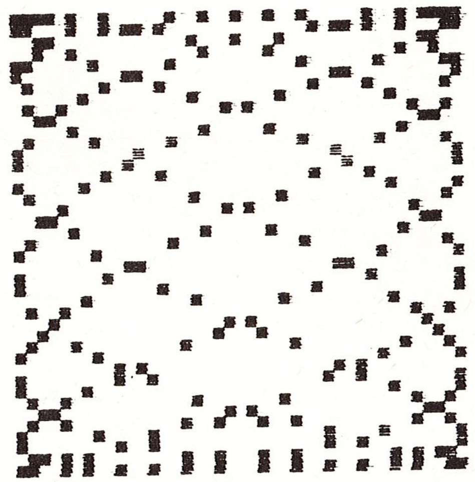
The delightful design produced on the screen can be used in a number of ways. One way is to compare two frequencies. If they are the same,

the program will draw a circle. If they are different, the number of points where the curve touches either the vertical or horizontal edge is the ratio of the two frequencies. If f_1 is the known frequency, then the unknown frequency (f_2) is equal to the number of times the curve touches the vertical edge of the confining box, multiplied by the known frequency.

Here are some examples of the effects the program can produce. Of course, they are far more attractive in color on the screen than the printouts suggest.







There are two versions of the program. The first one allows you to enter your own choice of STEP and X and Y frequencies. STEPs should be around 50 or more if the X and Y frequencies

are in single figures, and correspondingly more as the size of the frequencies increases. The higher the step number, the greater the resolution of the final figure.

Enter this program into your computer:

```
10 REM LISSAJOUS FIGURES
20 REM FRAZER MELTON
22 PI=3.141592
25 CLS
30 INPUT "STEPS";S
34 REM ENTER '0' TO END
35  IF S=0 THEN END
40 INPUT "Y FREQUENCY";Y
50 INPUT "X FREQUENCY";X
80 MODE (1)
82 Z=INT(RND(0)*3)+2
85 COLOR (Z)
90 FOR A=0 TO 2*PI STEP PI/S
100 SET ((20*SIN(A*Y)+60),(20*COS(A*X)+30))
110 NEXT A
120 FOR T=1 TO 2000:NEXT
130 RUN
```

Try the following sample values with this program:

STEP	Y FREQUENCY	X FREQUENCY
50	1	1
60	2	1
60	3	4
60	5	2
80	1	2.5
600	13	26.5
300	7	4
175	9	4

500	13	6	
500	4	12	- ABC
200	8	1	
250	3	8	
500	20	8	
500	15	20	

The next program chooses the two frequencies at random, between one and 20. The step size is fixed at 500, but there is no reason why you should not change that if you wish. There is a slight pause at the end of each figure and then the screen clears, and a new figure begins.

```
10 REM AUTO-LISSAJOUS
20 REM FRAZER MELTON
22 PI=3.141592
30 S=500
40 Y=INT(RND(0)*20)+1
50 X=INT(RND(0)*20)+1
80 MODE (1)
82 Z=INT(RND(0)*3)+2
85 COLOR (Z)
90 FOR A=0 TO 2*PI STEP PI/S
100 SET ((20*SIN(A*Y)+60),(20*COS(A*X)+30))
105 IF INKEY$<>"" THEN 40
110 NEXT A
120 FOR T=1 TO 2000:NEXT
130 GOTO 40
```

This final version is simply a modification of the preceding program, in which the screen does not clear between each image, so eventually you'll end up with a very complex overlay of designs.

```
10 REM AUTO-LISSAJOUS
20 REM FRAZER MELTON
21 REM MOVE LINE 80 TO 25 SO WILL NOT ERASE
22 PI=3.141592
25 MODE(1)
30 S=500
40 Y=INT(RND(0)*20)+1
50 X=INT(RND(0)*20)+1
82 Z=INT(RND(0)*3)+2
85 COLOR (Z)
90 FOR A=0 TO 2*PI STEP PI/S
100 SET ((20*SIN(A*Y)+60),(20*COS(A*X)+30))
105 IF INKEY$<>"" THEN 40
110 NEXT A
120 FOR T=1 TO 2000:NEXT
130 GOTO 40
```

As you know, the SET command is used to place dots of light on the screen at a position specified by the numbers which follow the command.

SET (X,Y) places the dot at location X (which can be zero to 127; the distance across the screen from left to right) and location Y (a number from zero to 63; the count down the screen).

The next program, MARTIAN LACE, uses SET to produce a balanced (and highly attractive) picture on the screen. As you'll see when you run it, the design evolves perpetually.

```
10 REM MARTIAN LACE
50 MODE(1)
60 X=INT(RND(0)*63)+1
70 Y=INT(RND(0)*31)+1
```

```
80 Z=INT(RND(0)*4)+1
85 IF RND(0)>.3 THEN Z=1
90 COLOR (Z)
100 SET (X,Y)
110 SET (128-X,Y)
120 SET (128-X,64-Y)
130 SET (X,64-Y)
160 GOTO 60
```

MARTIAN LACE is also used in the MAGIC COMPOSER program in the chapter on sound.

THE MOTION PICTURE MAN

You can use the computer as a kind of 'etch-a-sketch' toy in Mode 1. You enter a number (2, 3 or 4) to select the color you want, then use the cursor keys (the ones with arrows on them) to draw designs of your choice on the screen. The image begins in the top left hand corner of the screen.

As well as responding to the arrow keys, the program accepts, and acts on, the following commands:

C - copies picture created to some printers (such as the Seikosha GP-100 or GP-100A)

R - runs the program again from the beginning

N - moves the cursor back to the start position

E - freeze the picture

S - stop the program

```
10 REM MODE 1 SKETCHER
20 INPUT A
30 MODE(1)
40 COLOR A
50 X=0:Y=0
70 C$=INKEY$
80 IF C$="," AND X<127 THEN X=X+1
90 IF C$="M" AND X>0 THEN X=X-1
100 IF C$="." AND Y>0 THEN Y=Y-1
110 IF C$=" " AND Y<63 THEN Y=Y+1
120 SET (X,Y)
130 IF C$="C" THEN COPY
140 IF C$="R" THEN RUN
150 IF C$="N" THEN 50
160 IF C$="E" THEN 1000
170 IF C$="S" THEN END
180 GOTO 70
1000  GOTO 1000
```

CHAPTER FOUR — MAKING MUSIC

Although it may seem a little limited, the SOUND command on your computer is capable of producing some quite interesting effects.

The SOUND command is followed by two numbers. The first is the pitch (a number from 0 to 31, with 0 being a rest and 31 the highest note the computer can produce). The second is the duration (from 1 to 9, with 1 being the shortest).

We can use the computer to play all its notes with the following program, SOUND DEMO:

```
10 REM SOUND DEMO
20 FOR T=1 TO 31
30 SOUND T,1
40 NEXT
```

When you run this program, you'll hear all the notes played from the lowest A (note 1) to the highest D# (note 31).

COMPOSER ANDROID

The computer can also be programmed to choose notes at random and create some rather hopeless 'electronic music' as COMPOSER ANDROID demonstrates:

```
10 REM COMPOSER ANDROID
20 DURATION=INT(RND(0)*2+1)
30 PITCH=INT(RND(0)*31)+1
40 SOUND PITCH,DURATION
50 GOTO 20
```

As you'll hear, this is not very tuneful. It is possible, however, to write a program which sticks to one key (C) and manages to make noises which are much more deserving . the adjective 'musical'. Our next program, MAGIC COMPOSER shows this convincingly. You'll see that the program also creates a pretty design on the screen as the music unfolds. The design part of the program is explained in the previous chapter of this book, on page 39, where it is called MARTIAN LACE.

```
10 REM MAGIC COMPOSER
15 DIM A(11)
20 FOR Z=1 TO 11
30 READ A(Z)
40 NEXT Z
50 MODE(1)
60 X=INT(RND(0)*63)+1
70 Y=INT(RND(0)*31)+1
80 Z=INT(RND(0)*4)+1
90 COLOR (Z)
100 SET (X,Y)
110 SET (128-X,Y)
120 SET (128-X,64-Y)
130 SET (X,64-Y)
140 X=INT(RND(0)*11)+1
145 IF X=1 OR X>7 THEN M=2+INT(RND(0)*4) ELSE M=1
150 SOUND A(X),M
160 GOTO 60
1000 DATA 28,27,25,23,21,20,18,16,16,4,28
```

ORGAN

If you'd prefer to create your own music, rather than let the computer do it, you can use the next program, ORGAN, with which to unleash your creative powers.

When you first run the program, an input prompt will appear. In response to this question mark, enter a number between 1 and 9. The lower the number, the faster the 'organ' will play.

Here's the key to the notes you play.

KEY	NOTE
Z	C
X	D
C	E
V	F
B	G
N	A
M	B
,	C'
.	D'
space	E'

```
10 REM ORGAN
15 REM CHANGE A TO ALTER SPEED
20 INPUT A:CLS
30 C$=INKEY$
40 IF C$="Z" THEN SOUND 4,A
45 C$=INKEY$
50 IF C$="X" THEN SOUND 6,A
55 C$=INKEY$
60 IF C$="C" THEN SOUND 8,A
```

```
65 C$=INKEY$
70 IF C$="V" THEN SOUND 9,A
75 C$=INKEY$
80 IF C$="B" THEN SOUND 11,A
85 C$=INKEY$
90 IF C$="N" THEN SOUND 13,A
95 C$=INKEY$
100 IF C$="M" THEN SOUND 15,A
105 C$=INKEY$
110 IF C$="," THEN SOUND 16,A
115 C$=INKEY$
120 IF C$="." THEN SOUND 18,A
125 C$=INKEY$
130 IF C$=" " THEN SOUND 20,A
135 IF C$=":" THEN END
140 GOTO 30
```

This organ is quite flexible. Here's the 'music' of the folk song CLICK GO THE SHEARS to play on your organ. The music was transcribed by Peter Shaw:

```
C C X Z C B , , M N
B B N B C Z X X C X
C C X Z C B , , M N
. , M N B V C X Z , , ,
```

Chorus

```
. . , M . , space ,
N N M , N N B , C X
X C C C X Z C B , , , M N
N . , M N B V C X Z , , ,
```

CHAPTER FIVE — USING YOUR MEMORY

With the aid of the additional plug-in memory, your computer can be used as a 'real computer' in ways which were impossible (or, at best, inconvenient) on the unexpanded machine.

Let me explain what I mean by that. Before you bought your computer, you probably imagined it would be extremely useful as a means of storing and processing large amounts of information. There are several areas of practical use in both the home and small business in which personal computers have been employed. These include:

- maintaining details of a record collection
- looking after your personal library
- storing recipes
- maintaining personal financial records
- assisting in organising material for research projects

However, if you had tried to create long and complex programs to do these - and similar tasks - on your unexpanded computer, you may have discovered that once you'd entered the program to hold and process your data, there was so little memory left that the program was really only of use for demonstration purposes: "This program shows how the computer <u>could</u> act as a sorting and access aid for my record collection, <u>if</u> it had more memory".

When your computer was first released, there was little software available for it. You probably know that commercial software developers always keep a sharp eye on new computers. If it looks like a computer is going to take off, then it is worth writing and marketing software for the computer. There is a little of the 'chicken and egg' situation here. A wide variety of software packages available for a particular computer can help a potential purchaser make up his or her mind regarding which machine to buy.

You can now use your computer, with much of the software which is now available, as a cheap and very useful business or personal tool. There are some points to watch, however. You may well find that available programs suit your personal or business needs exactly. If that's the case, you just buy the program, load it in, and you're away. However, a program which works hand-in-glove with your present needs may, in the end, prove more of a handicap than a boon, as it may prevent you developing and expanding your activities as your needs change and develop.

How can you avoid this problem? One way is to use commercial software — or programs in magazines and books like this one — as the <u>starting point</u> for software tailored to your present, and anticipated future, needs. The other way, of course, is to write your own software from scratch, using the ideas you've learned by reading this book.

There are a number of areas where computers have found ready application in business, and your computer can certainly assist in this way with a small business. You'll find that, among other uses, you could develop programs to assist with the following:

- the payroll of a small firm can easily be managed with the aid of your computer

- keeping an up to date inventory of the stock of a small warehouse or shop is also possible, and provided that you choose 'codes' to represent particular items (rather than spelling out the name of each item in detail) you'll discover you can keep a surprisingly high number of different products in your inventory file

- accounts work (both accounts receivable and accounts payable), along with such things as forecasting budgetary trends, are well within the capability of your computer, even for quite sophisticated and complicated work

- word processing, after a fashion, can be managed on your computer

So, additional memory allows you to exercise your computer in ways which could have been unrealistic, or inconvenient on the unexpanded machine. But, it does not stop there. The extra memory gives you access to memory space to write far more elaborate, and satisfying, games than may have been possible before.

We'll look briefly at some ideas for such games at the end of this chapter, but firstly we'd like to mention another advantage of the additional memory; an advantage that applies equally to business and games programs. If you are constantly worrying about the possibility of running out of memory, you are unlikely to create your programs at your best.

The extra memory can be used, not only for making the program more elaborate and satisfying to use, but also in four important ways:

- You can include full instructions, and very clear user prompts, so the user knows exactly what he or she is expected to do in all cases

- Even if the instructions have been misunderstood, the extra memory can be used to advantage to add 'error traps' to prevent a mistake by the program user causing the entire program to come to a grinding halt

- The screen can be formatted in such a way as to reduce the chance of such an operator error occurring with, perhaps, the bottom two lines of the display being used only for user prompts, with these erased by two blank PRINT @ lines after the prompts have been followed

- The screen can also be formatted purely for visual effect, spreading the computer output around the entire screen to make it as easy as possible to read and understand it

These four suggestions apply - as we said - to any programs, whether they are for entertainment, instruction or business administration.

NASTY HABITS

After you've cut your programming teeth on short, simple programs, you're sure to want to start writing longer, more complex ones. When you do this, you may discover that many of the good habits you've learned will desert you.

If you're not worried about making a program tight and compact, you'll discover it is very easy to set up a long and sloppy set of IF/THENs which could easily have been replaced with a single IF/THEN sending action to a subroutine. When you have memory to spare, it often seems too much trouble to bother cleaning up your programs.

Discarded subroutines can clutter up the bottom ends of the listings, and GOTOs cover a multitude of situations which arose because you did not give sufficient thought to the highest line number you might need.

SURPRISES

One of the best things you can add to a longer program is the element of surprise. If you can include situations which do not occur every time a program is run, you'll be making sure that the game remains interesting for a much longer time than might be the case if every situation was triggered every time the game was run.

You may wish to use and adapt the simpler programs in this book, adding your own touches. Instructions, use of color and sound can all add to the value of the final program.

Another way of improving programs, and making them interesting to users for a longer time, it to use a feature you've no doubt seen in computer games before, the 'degree of difficulty'. When you allow the player to select a degree of difficulty, make sure that the user input really does alter the standard of play. Ensure that, even at the highest level of play, the final score (or successful landing on the moon, or a tally of obliterated aliens or whatever) is attainable in practice.

You could also add further interest to games by awarding points, or scores, or ratings and the like, which are genuinely related to the speed, skill, survival time or whatever the player demonstrated. A further twist is to award a 'rank' (like 'star fleet commander', 'novice' or 'incompetent fool') to the player, depending on how well he or she did. Points and ranks help to ensure that a player will remain interested in a game for a longer time than might otherwise be the case.

The use of a 'high score' is also a good idea. You can see high scores in a few programs in

this book such as MESOZOIC ATTACK.

After you've been writing programs for a while, you'll realize that many of them run too fast to be good games without the use of a 'delay subroutine' (such as an empty FOR/NEXT loop to slow things down). If the computer is meant to be thinking before responding to a move, it is more realistic for there to be a short pause before the computer replies, rather than come back instantly with its move or answer.

SPEED CONTROL

In contrast to this, you may well find that very long, moving-graphics programs run too slowly. You can ensure your program runs as fast as possible by starting with a GOSUB which jumps right over your main program to the end. In this subroutine, you assign variables, give the instructions, and so on.

When the computer comes across a GOSUB or GOTO command, it starts at the very first line of a program, then searches through, line by line, 'till it comes to the one it is looking for. Therefore, if an often-called subroutine is near the start of the listing, rather than near the end, the computer will find it more quickly than would be the case otherwise. The saving in time is not dramatic, but every fraction of a second you can save in a moving graphics program adds to its effectiveness.

IDEAS FOR YOUR OWN GAMES

The following list is by no means exhaustive, but should give you some ideas on the kind of programs you may be willing to tackle:

CRIBBAGE: Sir John Suckling, who lived from 1609 to 1642, is credited with the invention of this game, an elaboration of the older English game of NODDY. If you enjoy cribbage, you'll probably enjoy the challenge of creating an electronic opponent, which not only controls the cards, but also draws a computer representation of the cribbage board on the screen.

POKER: This, and many other card games, can be written using a deck of cards which is really the numbers one to 52, with one through 13 representing, say, the hearts, 14 through 26 the diamonds, and so on. Once you have a simple version of poker up and running, you may want to elaborate it, to play one of the many variations of the game, such as BLIND TIGER, WHISKEY POKER, STUD POKER, SEVEN-CARD STUD, LOW BALL and FREEZEOUT.

CHILDREN'S CARD GAMES: There is a wealth of games ideas in the field of children's card games and because they are generally much

simpler to play, with less subtle strategy, you'll probably find them easier to program than 'adult' card games.

REM: DM try to build a Prog "SNAP"

Children's card games include: SLAPJACK, OLD MAID, PIG, GO FISH and BEGGAR-YOUR-NEIGHBOUR.

rem: 5000 : Pyramid B

BACKGAMMON: Although possibly man's oldest board game, the rules were not recorded in a systematic form until the middle of the 1700's, when the famous game writer Edmond Hoyle decided to write them down. Since then, the only significant change in the game (and one which certainly adds to its appeal) is the provision of the doubling cube. You'll find that Backgammon may be somewhat simpler to write than other board games of pure skill, because the use of dice introduces an element of chance into the game from which the computer can benefit.

CHESS: While this may seem an impossible task at your present stage, you can approach it by trying to write a kind of 'mini-chess' you invent. Your version of chess could be on a six by six board, instead of the standard eight by eight, or could use just four pieces for each side. When you start thinking about it, you'll realise that chess (and other board games, for that matter) is a fruitful field to look for ideas to turn into games for your computer.

NINE MEN'S MORRIS: This game, named in Elizabethan times, is played on a simple board, but there is a wealth of strategy you can program into it.

CHECKERS: This, like Chess, can act as a source of ideas for variations. A 'mini-checkers'on a four by four board may be a good place to begin.

ROULETTE: The most famous gambling game of them all - Roulette - is a great one to computerise, and one which you should not find difficult to do. The relative ease of writing a Roulette program, compared to one which, say, plays Chess, comes from the fact that a Roulette program does not require any strategic 'intelligence', but simply has to accept bets and spin the wheel, then act on the result.

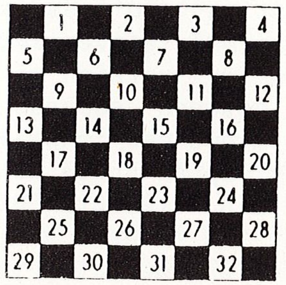
CHAPTER SIX — CREATING BOARD GAMES

This chapter of the book will lead you through the development of an almost-complete game of Checkers (or Draughts, as it is often called). The game is 'partially-complete' because it lacks the facility of multiple jumps. Apart from this, it will provide you with a lot of entertainment.

The main reason for including it in the book is so that a method for numbering boards for computer versions of board games can be explained. This system produces a board which the computer can manipulate relatively easily. A similar numbering system can be used for almost any board game, from Chess to Othello.

You should follow the text carefully, entering each version of the program as it is described. Then you'll have a pretty good understanding of how the system works. This information can then be used to create your own computer board games.

The normal way to number a checkers board is to count off the white squares from one to 32, as in this diagram:

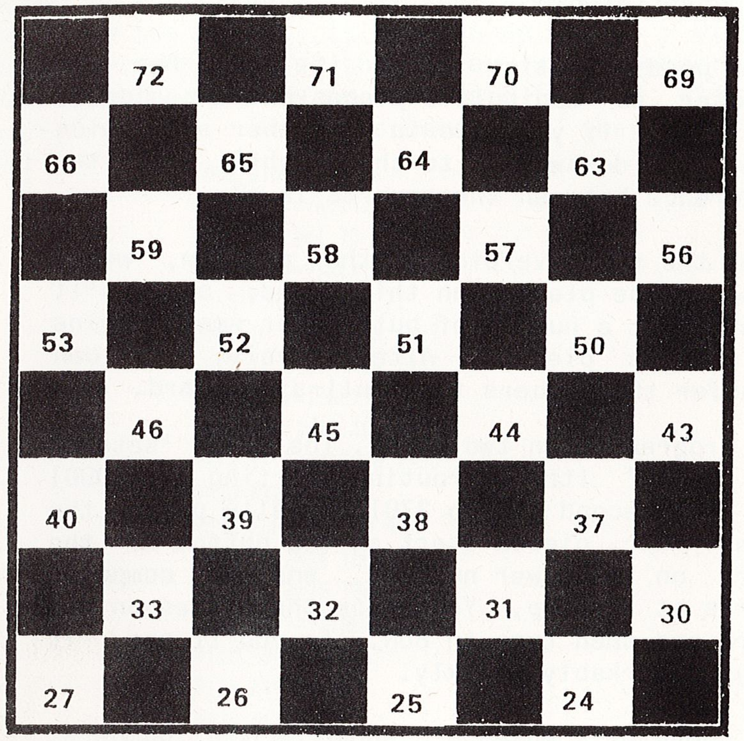


Actually, the black squares are counted, but white ones are used here because it is easier to see the numbers on white than on the black. When you look at the numbers on this board, you'll see the method creates a problem when we try to define a move in terms of the difference between two diagonally adjacent squares. In one direction, the difference between the squares can be three or four, and in the other direction, the difference can be five or four. There are also no 'spare' numbers to indicate where the edge of the board begins.

In an article in **Scientific American** in the 1960s, A L Samuels described a board-numbering system he had devised in which the difference between diagonally adjacent squares was always four and five (or minus four and minus five). The system also allowed for numbers to be given to squares which were 'off the board'. (You can read about Samuels' system in **Systems Analysis and Programming**, in READINGS FROM SCIENTIFIC AMERICAN, Strachey, C., W H Freeman and Co., San Francisco, 1971.)

Although Samuels' system worked, it was a little limiting, so I devised a system of my own which was more convenient for the computer to use. In my version of the system, the difference between squares is always six or seven (or minus six and minus seven). The system sets up an array of 82 elements, and allots certain elements of this array to squares on the board. All the others are understood by the computer to be off the board.

In this program, the computer allots a value of 9 to any square off the board, zero to an empty square, 1 to a computer's ordinary piece and 2



to its King, and -1 for the player's piece with -2 for the player's King. This may sound a little complicated, but as you read on, it should become clear.

Look at the numbered board above. You can see that if you move from the top right hand corner (69) to the square diagonally below it (63) the difference between the two squares is -6. Now, choose any other square on the board from which you can move down and to the left, and you'll see there is a difference of -6 between the square you started on, and the square on which you have finished.

This predictability makes it easy for the computer to manipulate pieces on the board. You'll see if you move in the other direction (that is downwards to the right) that the difference between the squares is -7.

With the first version of the program, we'll actually be playing on this board, so you'll have to get a number of buttons or small coins to use as pieces. Alternatively, you can transfer the numbers to a full-size board.

The program is in two parts. The first 'sets up the board' (the subroutine starting at 9000) and the second (10 to 370) actually plays the game. Your pieces start at the bottom of the board on the lower numbers, and the computer starts at the top. You place the pieces on the board and then type in RUN. As you'll see, it plays remarkably quickly.

The computer's moves are shown as two numbers. The first is the square it is moving from, and the second is the square into which it will move. Move the computer's piece on the board as instructed and then decide on your move. Make sure you actually move your piece on the board before you make the move, so you don't forget what the move was. Remember that there is no provision within the program for multiple jumps by either you or the computer. As well,

the game in this version cannot make or use Kings.

Here is the first part of the program, which sets up the board. It creates an array - DIM A(82) - then fills the array with numbers representing pieces on the board (1 and -1), empty squares (0) and squares off the board (9).

```
9000 DIM A(82)
9030 X(1)=-6:X(2)=-7
9050 FOR Z=1 TO 82
9060 A(Z)=9:NEXT
9065 FOR Z=56 TO 72
9066 IF Z=67 THEN Z=69
9067 IF Z=60 THEN Z=63
9070 A(Z)=1:NEXT
9075 FOR Z=43 TO 53
9076 IF Z=47 THEN Z=50
9080 A(Z)=0:NEXT
9090 FOR Z=24 TO 40
9095 IF Z=34 THEN Z=37
9096 IF Z=28 THEN Z=30
9100 A(Z)=-1:NEXT
9110 A$="MY MOVE"
9120 B$="YOURS"
9130 RETURN
```

Check to see if it is working properly before you proceed. RUN the program, and ignore the error message you get. Get the computer to print out the following, to make sure it has asssigned values correctly. The correct answer to each is given:

```
PRINT  A(23)   9
PRINT  A(54)   9
PRINT  A(64)   1
PRINT  A(38)   -1
PRINT  A(51)   0
PRINT  A(73)   9
```

Now enter the following:

```
1 REM CHECKERS
5 CLS:GOSUB 9000
10 Q=0
20 Z=24
25 IF A(Z)=9 OR A(Z)<1 THEN 100
50 X=1
60 IF A(Z+X(X))<0 AND A(Z+2*X(X))=0 THEN Q=X(X)
70 IF Q<>0 AND Z+2*Q THEN 120
80 Q=0
90 IF X<2 THEN X=X+1:GOTO 60
100 IF Z<72 THEN Z=Z+1:GOTO 25
110 IF Q=0 THEN 160
120 A(Z+Q)=0
130 A(Z+2*Q)=A(Z)
140 A(Z)=0
150 PRINT A$;Z;Z+2*Q
155 GOTO 320
160 Y=0
170 Z=INT(RND(0)*49)+24
180 Y=Y+1
190 IF A(Z)<>1 AND A(Z)<>2 THEN 170
200 X=1
210 IF A(Z+X(X))=0 THEN Q=X(X)
220 IF A(Z)=2 AND A(Z-X(X))=0 THEN Q=-X(X)
230 IF Q<>0 THEN 290
240 IF X<2 THEN X=X+1:GOTO 210
260 IF Y<100 THEN 170
270 PRINT "YOU WIN"
280 END
290 A(Z+Q)=A(Z)
```

```
300 A(Z)=0
310 PRINT @ 0,A$;Z;Z+Q
320 PRINT:PRINT B$;
330 INPUT A,B
340 CLS:A(B)=A(A)
350 A(A)=0
360 IF  ABS(A-B)>7 THEN A(((A+B)/2))=0
370 GOTO 10
```

Play a complete game with the computer, using the board provided, then return to the book, and we'll look at ways of improving the game. You move by entering the number of the square you're moving from, then a comma (,), then the square you're moving to, before pressing RETURN.

The next version of the program adds a board display. You still need to refer to the diagram to get the numbers to enter, and there is no provision for Kings. Delete line 320 of your program, and delete the CLS from line 340. Change 310 and 150 to GOSUB 5000, and add line 15. Change line 155 to GOTO 330.

```
1 REM CHECKERS
2 REM VERSION II
5 CLS:GOSUB 9000
10 Q=0
15 GOSUB 5000
20 Z=24
25 IF A(Z)=9 OR A(Z)<1 THEN 100
50 X=1
60 IF A(Z+X(X))<0 AND A(Z+2*X(X))=0 THEN Q=X(X)
70 IF Q<>0 AND Z+2*Q THEN 120
80 Q=0
90 IF X<2 THEN X=X+1:GOTO 60
```

```
100 IF Z<72 THEN Z=Z+1:GOTO 25
110 IF Q=0 THEN 160
120 A(Z+Q)=0
130 A(Z+2*Q)=A(Z)
140 A(Z)=0
150 GOSUB 5000
155 GOTO 330
160 Y=0
170 Z=INT(RND(0)*49)+24
180 Y=Y+1
190 IF A(Z)<>1 AND A(Z)<>2 THEN 170
200 X=1
210 IF A(Z+X(X))=0 THEN Q=X(X)
220 IF A(Z)=2 AND A(Z-X(X))=0 THEN Q=-X(X)
230 IF Q<>0 THEN 290
240 IF X<2 THEN X=X+1:GOTO 210
260 IF Y<100 THEN 170
270 PRINT "YOU WIN"
280 END
290 A(Z+Q)=A(Z)
300 A(Z)=0
310 GOSUB 5000
330 INPUT A,B
340 A(B)=A(A)
350 A(A)=0
360 IF  ABS(A-B)>7 THEN A(((A+B)/2))=0
370 GOTO 10
5000 FOR M=24 TO 72
5010 IF A(M)=1 THEN A(M)=66
5020 IF A(M)=-1 THEN A(M)=87
5030 IF A(M)=0 THEN A(M)=32
5050 NEXT M
5055 PRINT @ 0,"                    "
5060 IF Q<>0 THEN PRINT @ 0,A$;Z;Z+Q
5070 PRINT @ 320,"          "
5080 PRINT @ 64,;
```

```
5090 FOR K=0 TO 3
5100 FOR J=0 TO 3
5110 PRINT CHR$(128);CHR$(A(72-J-13*K));
5120 NEXT J
5130 PRINT
5140 FOR J=0 TO 3
5150 PRINT CHR$(A(66-J-13*K));CHR$(128);
5160 NEXT J
5170 PRINT
5180 NEXT K
5200 FOR M=24 TO 72
5210 IF A(M)=66 THEN A(M)=1
5220 IF A(M)=87 THEN A(M)=-1
5230 IF A(M)=32 THEN A(M)=0
5250 NEXT M
5260 RETURN

9000 DIM A(82)
9030 X(1)=-6:X(2)=-7
9050 FOR Z=1 TO 82
9060 A(Z)=9:NEXT
9065 FOR Z=56 TO 72
9066 IF Z=67 THEN Z=69
9067 IF Z=60 THEN Z=63
9070 A(Z)=1:NEXT
9075 FOR Z=43 TO 53
9076 IF Z=47 THEN Z=50
9080 A(Z)=0:NEXT
9090 FOR Z=24 TO 40
9095 IF Z=34 THEN Z=37
9096 IF Z=28 THEN Z=30
9100 A(Z)=-1:NEXT
9110 A$="MY MOVE"
9120 B$="YOURS"
9130 RETURN
```

The next version of the program adds Kings (the computer's King is a K, yours is the hash, #, symbol), and with the addition of a single line, 215, adds an element of foresight (the computer will be reluctant to move into danger on a non-capture move) which markedly improves the computer's play.

```
1 REM CHECKERS
2 REM VERSION III
3 REM KINGS ADDED
4 REM PLUS SOME STRATEGY
5 CLS:GOSUB 9000
10 Q=0
15 GOSUB 5000
20 Z=24
25 IF A(Z)=9 OR A(Z)<1 THEN 100
50 X=1
55 IF Z<71 THEN IF A(Z-X(X))<0 AND A(Z-2*X(X))=0 THEN Q=-X(X)
60 IF A(Z+X(X))<0 AND A(Z+2*X(X))=0 THEN Q=X(X)
70 IF Q<>0 AND Z+2*Q THEN 120
80 Q=0
90 IF X<2 THEN X=X+1:GOTO 60
100 IF Z<72 THEN Z=Z+1:GOTO 25
110 IF Q=0 THEN 160
120 A(Z+Q)=0
130 A(Z+2*Q)=A(Z)
140 A(Z)=0
150 GOSUB 5000
155 GOTO 330
160 Y=0
170 Z=INT(RND(0)*49)+24
180 Y=Y+1
190 IF A(Z)<>1 AND A(Z)<>2 THEN 170
200 X=1
210 IF A(Z+X(X))=0 THEN Q=X(X)
215 IF A(Z+2*X(X))<0 AND Q<>0 AND RND(0)>.7 THEN Q=0
```

```
220 IF A(Z)=2 AND A(Z-X(X))=0 THEN Q=-X(X)
230 IF Q<>0 THEN 290
240 IF X<2 THEN X=X+1:GOTO 210
260 IF Y<100 THEN 170
270 PRINT "YOU WIN"
280 END
290 A(Z+Q)=A(Z)
300 A(Z)=0
310 GOSUB 5000
330 INPUT A,B
340 A(B)=A(A)
350 A(A)=0
360 IF  ABS(A-B)>7 THEN A(((A+B)/2))=0
370 GOTO 10
5000 FOR M=24 TO 72
5005 IF A(M)=1 AND M>23 AND M<28 THEN A(M)=2
5006 IF A(M)=-1 AND M>68 THEN A(M)=-2
5010 IF A(M)=1 THEN A(M)=66
5015 IF A(M)=2 THEN A(M)=75
5020 IF A(M)=-1 THEN A(M)=87
5025 IF A(M)=-2 THEN A(M)=35
5030 IF A(M)=0 THEN A(M)=32
5050 NEXT M
5055 PRINT @ 0,"                    "
5060 IF Q<>0 THEN PRINT @ 0,A$;Z;Z+Q
5070 PRINT @ 320,"         "
5080 PRINT @ 64,;
5090 FOR K=0 TO 3
5100 FOR J=0 TO 3
5110 PRINT CHR$(128);CHR$(A(72-J-13*K));
5120 NEXT J
5130 PRINT
5140 FOR J=0 TO 3
5150 PRINT CHR$(A(66-J-13*K));CHR$(128);
5160 NEXT J
5170 PRINT
5180 NEXT K
```

```
5200 FOR M=24 TO 72
5210 IF A(M)=66 THEN A(M)=1
5215 IF A(M)=75 THEN A(M)=2
5220 IF A(M)=87 THEN A(M)=-1
5225 IF A(M)=35 THEN A(M)=-2
5230 IF A(M)=32 THEN A(M)=0
5250 NEXT M
5260 RETURN
9000 DIM A(82)
9030 X(1)=-6:X(2)=-7
9050 FOR Z=1 TO 82
9060 A(Z)=9:NEXT
9065 FOR Z=56 TO 72
9066 IF Z=67 THEN Z=69
9067 IF Z=60 THEN Z=63
9070 A(Z)=1:NEXT
9075 FOR Z=43 TO 53
9076 IF Z=47 THEN Z=50
9080 A(Z)=0:NEXT
9090 FOR Z=24 TO 40
9095 IF Z=34 THEN Z=37
9096 IF Z=28 THEN Z=30
9100 A(Z)=-1:NEXT
9110 A$="MY MOVE"
9120 B$="YOURS"
9130 RETURN
```

Our final version of the program adds a few features, such as a 'score' which counts how many pieces each of you have captured. And in this version, you can forget the numbered board. The board now appears on the screen with the letters ABCDEFGH across the top, and the numbers 1 to 8 down the side. You make a move simply by entering the co-ordinates of the relevant squares (such as E6,C4) instead of the

numbers and the computer interprets them to make your desired move.

Before I give you the listing of the final version, here's a printout of part of a game played against it. When you see it on the screen, the dots will be replaced by black squares, and the 'less than' symbols down the right hand edge will be replaced by a black and white line.

```
ABCDEFGH
.B.B.B.B< 1
B.B.B.B.< 2
.B.B. .B< 3
 . .B. .< 4
. . . . < 5
W.W.W.W.< 6
.W.W.W.W< 7
W.W.W.W.< 8
? G6,H5
```

```
ABCDEFGH
.B.B.B.B< 1
B.B.B.B.< 2
. .B. .B< 3
B. .B. .< 4
. . . .W< 5
W.W.W. .< 6
.W.W.W.W< 7
W.W.W.W.< 8
? F7,G6
```

```
ABCDEFGH
.B.B.B.B< 1
B.B. .B.< 2
. .B. . < 3
B. .B.B.< 4
. . . . < 5
W.W.W.W.< 6
.W.W. .W< 7
W.W.W.W.< 8
? G8,F7

      MY SCORE IS 1
    YOUR SCORE IS 1
```

```
ABCDEFGH
.B.B.B.B< 1
B.B. . .< 2
. .B.B. < 3
B. .B.B.< 4
. . . . < 5
W.W.W.W.< 6
.W.W.W.W< 7
W.W.W. .< 8
? G6,H5

          MY SCORE IS 1
        YOUR SCORE IS 1
```

```
ABCDEFGH
.B.B.B.B< 1
B.B. . .< 2
. .B.B. < 3
B. . .B.< 4
. .B. .W< 5
W.W.W. .< 6
.W.W.W.W< 7
W.W.W. .< 8
? C6,E4

          MY SCORE IS 1
        YOUR SCORE IS 1
```

```
ABCDEFGH
.B.B.B.B< 1
B.B. . .< 2
. .B. . < 3
B. . .B.< 4
. .B. .W< 5
W. .W. .< 6
.W.W.W.W< 7
W.W.W. .< 8
? E6,C4
```

```
ABCDEFGH
.B.B.B.B< 1
B.B. . .< 2
. . . . < 3
B. . .B.< 4
.B. . .W< 5
W. . . .< 6
.W.W.W.W< 7
W.W.W. .< 8
? A6,C4

          MY SCORE IS 3
        YOUR SCORE IS 3
```

And here's the final version of our Checkers game:

```
       1 REM CHECKERS
       2 REM VERSION IV
       3 REM PROPER MOVE INPUT
       4 REM PLUS ON-SCREEN SCORING
210    5 CLS:GOSUB 9000
215    10 Q=0
220    15 GOSUB 5000
225    20 Z=24
230    25 IF A(Z)=9 OR A(Z)<1 THEN 100
35     50 X=1
40     55 IF Z<71 THEN IF A(Z-X(X))<0 AND A(Z-2*X(X))=0 THEN Q=-X(X)
245    60 IF A(Z+X(X))<0 AND A(Z+2*X(X))=0 THEN Q=X(X)
250    70 IF Q<>0 AND Z+2*Q THEN 120
255    80 Q=0
260 →  90 IF X<2 THEN X=X+1:GOTO ~~60~~ (245)
65     100 IF Z<72 THEN Z=Z+1:GOTO 25(230)
70     110 IF Q=0 THEN 160 (310)
75     120 A(Z+Q)=0
80     130 A(Z+2*Q)=A(Z)
85     135 CS=CS+1
90     136 PRINT @ 392,"MY SCORE IS"CS      — NOT IN
95     140 A(Z)=0
300    150 GOSUB 5000
```

```
305  155 GOTO 330
310  160 Y=0
315  170 Z=INT(RND(0)*49)+24
320  180 Y=Y+1
325  190 IF A(Z)<>1 AND A(Z)<>2 THEN 170
330  200 X=1
335  210 IF A(Z+X(X))=0 THEN Q=X(X)
340  215 IF A(Z+2*X(X))<0 AND Q<>0 AND RND(0)>.7 THEN Q=0
345  220 IF A(Z)=2 AND A(Z-X(X))=0 THEN Q=-X(X)
350  230 IF Q<>0 THEN 290(375)
355  240 IF X<2 THEN X=X+1:GOTO 210 (335)
360  260 IF Y<100 THEN 170 (315)
365  270 PRINT "YOU WIN"
370  280 END
375  290 A(Z+Q)=A(Z)
380  300 A(Z)=0
385  310 GOSUB 5000
390  330 GOSUB 1000
395  340 A(B)=A(A)
400  350 A(A)=0
405  360 IF  ABS(A-B)>7 THEN A(((A+B)/2))=0:HS=HS+1
NOT IN  →410  365 IF HS<>0 THEN PRINT @ 422,"YOUR SCORE IS";HS
415  370 GOTO 10
500  1000 INPUT V$,N$
505  1010 FOR U=1 TO 32
510  1020 IF Y$(U)=V$ THEN A=R(U)
515  1030 IF Y$(U)=N$ THEN B=R(U)
520  1040 NEXT
525  1050 RETURN
     5000 FOR M=24 TO 72
     5005 IF A(M)=1 AND M>23 AND M<28 THEN A(M)=2
     5006 IF A(M)=-1 AND M>68 THEN A(M)=-2
     5010 IF A(M)=1 THEN A(M)=66        (B)
     5015 IF A(M)=2 THEN A(M)=75        (K)
     5020 IF A(M)=-1 THEN A(M)=87       (W)
     5025 IF A(M)=-2 THEN A(M)=35        #
```

```
             5030 IF A(M)=0 THEN A(M)=32
             5050 NEXT M
NOT IN   —   5070 PRINT @ 320,"        "
NOT IN   →   5080 PRINT @ 64,;
             5090 FOR K=0 TO 3
             5100 FOR J=0 TO 3
             5110 PRINT CHR$(128);CHR$(A(72-J-13*K));
             5120 NEXT J
             5130 PRINT
NOT IN       5140 FOR J=0 TO 3
             5150 PRINT CHR$(A(66-J-13*K));CHR$(128);
             5160 NEXT J
             5170 PRINT
             5180 NEXT K
             5200 FOR M=24 TO 72
             5210 IF A(M)=66 THEN A(M)=1
             5215 IF A(M)=75 THEN A(M)=2
             5220 IF A(M)=87 THEN A(M)=-1
             5225 IF A(M)=35 THEN A(M)=-2
             5230 IF A(M)=32 THEN A(M)=0
             5250 NEXT M
             5260 RETURN
             9000 DIM A(82)
             9030 X(1)=-6:X(2)=-7
             9050 FOR Z=1 TO 82
             9060 A(Z)=9:NEXT
             9065 FOR Z=56 TO 72
             9066 IF Z=67 THEN Z=69
             9067 IF Z=60 THEN Z=63
             9070 A(Z)=1:NEXT
             9075 FOR Z=43 TO 53
             9076 IF Z=47 THEN Z=50
             9080 A(Z)=0:NEXT
             9090 FOR Z=24 TO 40
             9095 IF Z=34 THEN Z=37
             9096 IF Z=28 THEN Z=30
```

```
9100 A(Z)=-1:NEXT
9110 A$="MY MOVE"
9120 B$="YOURS"
9130 FOR J=72 TO 296 STEP 32
9140 PRINT @ J,CHR$(133);INT(J/32)-1
9150 NEXT J
9160 PRINT @ 32,"ABCDEFGH"
9170 PRINT @ 64,">PLEASE":PRINT">>STAND":PRINT">>>BY"
9180 DIM R(32),Y$(32)
9190 FOR T=1 TO 32
9200 READ X$:READ Q
9230 R(T)=Q:Y$(T)=X$
9240 NEXT
9250 HS=0:CS=0
9500 RETURN
9510 DATA "B1",72,"D1",71,"F1",70,"H1",69
9520 DATA "A2",66,"C2",65,"E2",64,"G2",63
9530 DATA "B3",59,"D3",58,"F3",57,"H3",56
9540 DATA "A4",53,"C4",52,"E4",51,"G4",50
9550 DATA "B5",46,"D5",45,"F5",44,"H5",43
9560 DATA "A6",40,"C6",39,"E6",38,"G6",37
9570 DATA "B7",33,"D7",32,"F7",31,"H7",30
9580 DATA "A8",27,"C8",26,"E8",25,"G8",24
```

CHAPTER SEVEN — A GAMUT OF GAMES

The 'lessons' are now behind you. In this next section of the book we have a number of games which I am sure you'll enjoy playing.

Once you have them up and running on your system, spend a bit of time adding your own personal touch to them. Modify them in any way you like. As well, studying the program listings should give a number of additional ideas to include in your own programs.

The games in this section are:

LAUNCH PAD
LEAPFROG
NUMBER CAVALCADE
ROMTHELLO
LIFE
NIMGRAB
LAS VEGAS HIGH
SWITCHEROO
MANCALA

Good gameplaying.

LAUNCH PAD

Here's a simple program which fills your TV with 'rocket ships' blasting off up the screen. You may well be able to work this very effective display into a game.

Here's an indication of what it looks like on screen:

```
              A
             I^I
            //:\\
             /*\
                                       A
                                      I^I
                                     //:\\
                                      /*\
                  A
                 I^I
                //:\\
                 /*\
                               A
                              I^I
                             //:\\
                              /*\
        A
       I^I
      //:\\
       /*\
                                 A
                                I^I
                               //:\\
                                /*\
```

And here's the brief listing:

```
10 REM LAUNCH PAD
15 DIM A$(5)
17 CLS
20 A$(1)="   A"
30 A$(2)="  I^I"
40 A$(3)=" //:\\"
50 A$(4)="  /*\"
130 ~~90~~ Q=INT(RND(0)*25)+1
132 ~~110~~ FOR R=1 TO 4
134 ~~130~~ PRINT TAB(Q);A$(R)
136 ~~135~~ NEXT R
140 FOR P=1 TO Q/3
150 PRINT
160 NEXT P
170 GOTO 20
```

LEAPFROG

Here's a game which will test your wits. The game begins with a display like this:

```
      THAT WAS MOVE  0

      $ $ $ $   * * * *
      1 2 3 4 5 6 7 8 9

    WHICH PIECE TO MOVE?
```

The aim of the game is to end up with the asterisks on the left and the dollar signs on the right. You can only move into the empty space, either by sliding sideways into it, or by jumping over one piece (hence the name of the game) into the space.

The aim of the game is to get the pieces swapped in the fewest possible moves. You'll find it a difficult, frustrating challenge.

```
10 REM LEAPFROG
20 X=-1:M=9
30 CLS
90 FOR Q=1 TO M
100 IF Q<5 THEN A(Q)=ASC("$")
110 IF Q>5 THEN A(Q)=ASC("*")
120 NEXT Q
130 FOR T=1 TO 400:NEXT T
140 C=0
150 X=X+1
170 PRINT @ 32,"THAT WAS MOVE ";X
180 PRINT:PRINT
200 GOSUB 1000
210 PRINT:PRINT
```

```
220 PRINT "YOU HAVE";C;"CORRECT"
230 IF C=8 THEN 360
240 GOSUB 320
270 GOTO 130
320 PRINT @ 390,"WHICH PIECE TO MOVE?"
324 IF INKEY$<>"" THEN 324
325 A$=INKEY$
326 IF A$<"1" OR A$>"9" THEN 325
327 T=VAL(A$)
328 PRINT @ 390,"                    "
330 K=A(T)
340 A(T)=0:A(H)=K
350 RETURN
360 PRINT:PRINT TAB(8);"WELL DONE!"
370 PRINT:PRINT TAB(4);"YOU SOLVED IT IN","      JUST";X;"MOVES"
1000 PRINT:PRINT
1010 FOR Z=1 TO 9
1020 IF A(Z)<>0 THEN PRINT CHR$(A(Z));" ";
1030 IF A(Z)=0 THEN PRINT "  ";:H=Z
1040 IF Z<5 AND A(Z)=ASC("*") THEN C=C+1
1050 IF Z>5 AND A(Z)=ASC("$") THEN C=C+1
1070 NEXT Z
1080 PRINT:PRINT "1 2 3 4 5 6 7 8 9"
1090  RETURN
```

```
THAT WAS MOVE  22



$ *   * $ $ * * $
1 2 3 4 5 6 7 8 9


YOU HAVE 4 CORRECT
```

NUMBER CAVALCADE

In this simple game, you think of a number between one and 63. The computer then proceeds to show you a series of groups of numbers. If the number you're thinking of is within a particular group, you press the "Y" (for 'yes') key. If your number is not on the screen, you press the "N" key.

At the end of the game, the computer will tell you the number you had thought of.

```
1 3 5 7 9 11 13 15 17
19 21 23 25 27 29 31
33 35 37 39 41 43 45
47 49 51 53 55 57 59
61 63

    Y

 2 3 6 7 10 11 14 15
18 19 22 23 26 27 30
31 34 35 38 39 42 43
46 47 50 51 54 55 58
59 62 63

    N

 4 5 6 7 12 13 14 15
20 21 22 23 28 29 30
31 36 37 38 39 44 45
46 47 50 51 54 55 58
61 62 63

    Y
```

```
8 9 10 11 12 13 14 15
24 25 26 27 28 29 30
31 40 41 42 43 44 45
46 47 56 57 58 59 60
61 62 63

    Y

16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31 48 49 50 51 52
53 54 55 56 57 58 59
60 61 62 63

    Y

32 33 34 35 36 37 38
39 40 41 42 43 44 45
46 47 48 49 50 51 52
53 54 55 56 57 58 59
60 61 62 63

    N

    YOUR NUMBER WAS 29
```

```
10 REM NUMBER CAVALCADE
15 X=.5:N=0:GOSUB 1020
20 PRINT "1 3 5 7 9 11 13 15 17"
30 PRINT "19 21 23 25 27 29 31"
40 PRINT "33 35 37 39 41 43 45"
50 PRINT "47 49 51 53 55 57 59"
60 PRINT "61 63"
70 GOSUB 1000
80 PRINT " 2 3 6 7 10 11 14 15"
90 PRINT "18 19 22 23 26 27 30"
100 PRINT "31 34 35 38 39 42 43"
```

```
110 PRINT "46 47 50 51 54 55 58"
120 PRINT "59 62 63"
130 GOSUB 1000
140 PRINT " 4 5 6 7 12 13 14 15"
150 PRINT "20 21 22 23 28 29 30"
160 PRINT "31 36 37 38 39 44 45"
170 PRINT "46 47 50 51 54 55 58"
180 PRINT "61 62 63"
190 GOSUB 1000
200 PRINT "8 9 10 11 12 13 14 15"
210 PRINT "24 25 26 27 28 29 30"
220 PRINT "31 40 41 42 43 44 45"
230 PRINT "46 47 56 57 58 59 60"
240 PRINT "61 62 63"
250 GOSUB 1000
260 PRINT "16 17 18 19 20 21 22"
270 PRINT "23 24 25 26 27 28 29"
280 PRINT "30 31 48 49 50 51 52"
290 PRINT "53 54 55 56 57 58 59"
300 PRINT "60 61 62 63"
400 GOSUB 1000
410 PRINT "32 33 34 35 36 37 38"
420 PRINT "39 40 41 42 43 44 45"
430 PRINT "46 47 48 49 50 51 52"
440 PRINT "53 54 55 56 57 58 59"
450 PRINT "60 61 62 63"
460 GOSUB 1000
470 PRINT "YOUR NUMBER WAS";N
480 GOTO 480
1000 X=X+X
1005 GOSUB 1060
1010 SOUND (2*X/3+2),1
1020 FOR T= 1 TO 500:NEXT T
1030 CLS
1040 PRINT:PRINT:PRINT
1050 RETURN
```

```
1060 IF INKEY$<>""  THEN 1060
1065 A$=INKEY$
1070 IF A$<>"N" AND A$<>"Y" THEN 1065
1080 IF A$="Y" THEN N=N+X
1090 RETURN
```

ROMTHELLO

This major program is based on the board game known either as REVERSI or OTHELLO. Invented in the late eighteen hundreds, it is played on an ordinary eight by eight board. When played on a board, you use pieces which have different colors on each side. The game begins with four pieces placed on the central squares.

From this point on, you move by placing one of your pieces next to a computer piece or pieces, with another of your pieces further on. When that happens, all the computer pieces 'reverse' to become your pieces.

Here's how it works. Suppose a line of pieces looked like this:

```
OXXXX
```

If you decided to put your piece (an O) at the end of the line like this:

```
OXXXXO
```

The computer pieces (the X's) would reverse, so the line looked like this after your move:

```
OOOOOO
```

The game continues until every square on the board is filled, or neither player can move. As you can imagine from the sample move shown above, fortunes change swiftly in this game, as rows branching off your position, such as on

the diagonals, can be changed with a single move.

If you cannot move at any time, you signal this to the computer by entering a zero.

This sample run shows the early stages of one game played against the computer:

```
    1 2 3 4 5 6 7 8
 1  . . . . . . . .  1
 2  . . . . . . . .  2
 3  . . . . . . . .  3
 4  . . . X O . . .  4
 5  . . . O X . . .  5
 6  . . . . . . . .  6
 7  . . . . . . . .  7
 8  . . . . . . . .  8
    1 2 3 4 5 6 7 8

COMPUTER  2     HUMAN  2

    1 2 3 4 5 6 7 8
 1  . . . . . . . .  1
 2  . . . . . . . .  2
 3  . . . . . . . .  3
 4  . . . X O . . .  4
 5  . . . X X . . .  5
 6  . . . X . . . .  6
 7  . . . . . . . .  7
 8  . . . . . . . .  8
    1 2 3 4 5 6 7 8

COMPUTER  4     HUMAN  1
```

```
     1 2 3 4 5 6 7 8
 1   . . . . . . . .   1
 2   . . . . . . . .   2
 3   . . . . . . . .   3
 4   . . . X O . . .   4
 5   . . . O X . . .   5
 6   . . O X . . . .   6
 7   . . . . . . . .   7
 8   . . . . . . . .   8
     1 2 3 4 5 6 7 8

COMPUTER  3      HUMAN  3

     1 2 3 4 5 6 7 8
 1   . . . . . . . .   1
 2   . . . . . . . .   2
 3   . . . . . . . .   3
 4   . . . X O . . .   4
 5   . . X X X . . .   5
 6   . . O X . . . .   6
 7   . . . . . . . .   7
 8   . . . . . . . .   8
     1 2 3 4 5 6 7 8

COMPUTER  5      HUMAN  2

     1 2 3 4 5 6 7 8
 1   . . . . . . . .   1
 2   . . . . . . . .   2
 3   . . . . . . . .   3
 4   . . . X O . . .   4
 5   . . X X O . . .   5
 6   . . O O O . . .   6
 7   . . . . . . . .   7
 8   . . . . . . . .   8
     1 2 3 4 5 6 7 8
```

```
     1 2 3 4 5 6 7 8
 1   . . . . . . . .   1
 2   . . . . . . . .   2
 3   . . X . . . . .   3
 4   . O O O O . . .   4
 5   . . O O X . . .   5
 6   . X X O O X . .   6
 7   . . . . O . . .   7
 8   . . . . . . . .   8
     1 2 3 4 5 6 7 8

COMPUTER  5      HUMAN  9

     1 2 3 4 5 6 7 8
 1   . . . . . . . .   1
 2   . . . . . . . .   2
 3   . . X . . . . .   3
 4   . O O O O . . .   4
 5   . . O O X . . .   5
 6   . X X O X X . .   6
 7   . . . . X . . .   7
 8   . . . . X . . .   8
     1 2 3 4 5 6 7 8

COMPUTER  8      HUMAN  7

     1 2 3 4 5 6 7 8
 1   . . . . . . . .   1
 2   . . . . . . . .   2
 3   . . X . . . . .   3
 4   . O O O O . . .   4
 5   . . O O X . . .   5
 6   . X X O O O O .   6
 7   . . . . X . . .   7
 8   . . . . X . . .   8
     1 2 3 4 5 6 7 8
     1 2 3 4 5 6 7 8
1    . . . . . . . .  1
2    . . . . . . . .  2
3    . . X . . . . .  3
4    . O O O O . . .  4
5    . . O O X . . .  5
6    . X X X X X X X  6
7    . . . . X . . .  7
8    . . . . X . . .  8
     1 2 3 4 5 6 7 8

COMPUTER  11      HUMAN  6
```

And here's the listing of ROMTHELLO:

```
10 REM ROMTHELLO
20 GOTO 740
30 PRINT @ 384,"MY MOVE"
40 S=Z:T=X:H=0
50 FOR A=2 TO 9:FOR B=2 TO 9
60 IF A(A,B)<>46 THEN 210
70 Q=0
80 FOR C=-1 TO 1:FOR D=-1 TO 1
90 K=0:F=A:G=B
100 IF A(F+C,G+D)<>S THEN 130
110 K=K+1:F=F+C:G=G+D
120 GOTO 100
130 IF A(F+C,G+D)<>T THEN 150
140 Q=Q+K
150 NEXT D:NEXT C
160 IF A=2 OR A=9 OR B=2 OR B=9 THEN Q=Q*2
170 IF A=3 OR A=8 OR B=3 OR B=8 THEN Q=Q/2
180 IF (A=2 OR A=9) AND (B=3 OR B=8) THEN Q=Q/2:GOTO 190
185 IF (A=3 OR A=8) AND (B=2 OR B=9) THEN Q=Q/2
190 IF Q<H OR (RND(0)<.3 AND Q=H) THEN 210
200 H=Q:M=A:N=B
```

```
210 NEXT B:NEXT A
220 IF H=0 AND R=0 THEN 690
230 IF H=0 THEN 250
240 GOSUB 580
250 GOSUB 370
255 PRINT @ 384,"        "
260 PRINT @ 354, "ENTER YOUR MOVE"
270 PRINT:INPUT R
275 PRINT @ 354, "                    "
280 S=X:T=Z
290 IF R=0 THEN 350
300 IF R<11 OR  R>88 THEN 260
310 R=R+11
320 M=INT(R/10)
330 N=R-10*M
340 GOSUB 580
350 GOSUB 370
360 GOTO 30
370 REM PRINT BOARD
380 C=0:H=0
390 SOUND RND(0)*30,1
395 SOUND RND(0)*30,1
400 PRINT @ 30,"      1 2 3 4 5 6 7 8"
450 FOR B=2 TO 9:PRINT B-1;" ";
460 FOR D=2 TO 9
470 PRINT CHR$(A(B,D));" ";
480 IF A(B,D)=X THEN C=C+1
490 IF A(B,D)=Z THEN H=H+1
500 NEXT D
510 PRINT B-1
520 NEXT B
530 PRINT "    1 2 3 4 5 6 7 8"
540 PRINT
550 PRINT @ 480,"COMPUTER ";C;"   HUMAN ";H
570 RETURN
580 FOR C=-1 TO 1:FOR D=-1 TO 1
590 F=M:G=N
```

```
600 IF A(F+C,G+D)<>S THEN 630
610 F=F+C:G=G+D
620 GOTO 600
630 IF A(F+C,G+D) <> T THEN 670
640 A(F,G)=T
650 IF M=F AND N=G THEN 670
660 F=F-C:G=G-D:GOTO 640
670 NEXT D:NEXT C
680 RETURN
690 GOSUB 370
700 IF C>H THEN PRINT "I'M THE CHAMP!"
710 IF H>C THEN PRINT "YOU'RE THE CHAMP"
720 IF H=C THEN PRINT "IT'S A DRAW"
730 END
740 CLS
750 X=88:Z=48
760 DIM A(10,10)
770 FOR B=1 TO 10:FOR C=1 TO 10
780 IF B<>1 AND C<>1 AND B<>10 AND C<>10 THEN A(B,C)=46
790 NEXT C:NEXT B
800 A(5,5)=X:A(6,6)=X
810 A(6,5)=Z:A(5,6)=Z
820 P=0
830 GOSUB 370
840 GOTO 30
```

LIFE

There are some 'classic' computer programs, ones which found widespread acceptance the moment they were announced. In the classics category I would place HUNT THE WUMPUS, ELIZA and LIFE.

LIFE was invented by John Conway, of Cambridge University, late in 1970. The program simulates the birth, death and growth of cells in a closed colony. Rather than breeding indiscriminately like fruit flies until all available space and food is consumed, the entities within the colonies of LIFE proceed in a most orderly fashion.

The entities live on a grid (measuring 15 by 15, in this case) in accordance with the following rules devised by Mr Conway:

- every cell has eight neighbours, and the state of these eight determines the fate of that cell in the next generation

- if a cell has two, or three, neighbours (no more, no less) it will survive into the next generation

- if there are three surrounding cells, and the place on the grid being checked is empty, a new cell will be born in that cell in the next generation

- if there are four or more neighbours, the cell will not survive to the next generation

With the aid of these simple rules, some very startling colony patterns evolve. Some die off very quickly, others settle down into a single shape, or into two or three shapes that cycle in subsequent generations.

Here are a few generations from one run of CONWAY'S COLONY, as we've called our version of LIFE:

```
-----------------------------------------------------
                GENERATION  1

* * * *       * * * * * *
* *       *   *   * *   *
* * * * * *     *   * *
* * *   * *     * *     *
*       * *     * * * * *
* *     * * *       * * *
  *       * *   *     *
* * *   * * *   * * * * *
* * * *     * * * *   * *
    * * * * *   * * * *
* * * * * *   * * *     *
*     * *     *   *   * *
* * * *             * *
-----------------------------------------------------
```

```
                GENERATION  2

*   *       * *         *
          *   *         *
              *         *
            * *         *
    *         * *
* *             *
      *         *
        *
*

*                       *
          * * *   *     *
* * * * *           * * *
-----------------------------------------------------
                GENERATION  3

            * *
              * *     * *
              * *     * *
            *
  *         *   *
  * *           * *




            *
*   * * * * *       *   *
  * * * * * *       * * *
```

```
         GENERATION  4

           * * *
           *   *     * *
           *   *     * *
 * *           * *
 * *         * * *



    * *   *
            *     *   *
*         *       *   *
```

```
         GENERATION  5

            *
          *   *     * *
                    * *
              *
* *
* *         *   *
              *





        * * *
```

```
         GENERATION  6

            *
            *       * *
            *       * *

* *           *
* *           *
              *



          *
          *
          *
```

You'll see in the last generation there are three lines each of three cells. These are called 'traffic lights' because they go from:

```
*              *
*  to  * * *  to  *  to  * * *   and so on...
*              *
```

You can see the first traffic light (as * * *) at the bottom of generation five. The groups of four are stable units, which stay as they are from generation to generation, unless other cells touch the group from the outside. There are a number of shapes such as these which often occur in LIFE programs. You're likely to become familiar with them as you continue to run the program.

As you can see (lines 80 to 140) the original colony is generated at random. Once you've become familiar with this version of the program, you might like to try and rewrite it so that you can enter your own colony. You'll find when you do this that a balanced pattern, such as a couple of diamonds, is more likely to evolve in an interesting way than are the purely random colonies.

```
10 REM CONWAY'S COLONY
20 CLS
30 COLOR 2,1
50 DIM A(15,15)
60 DIM B(15,15)
70 G=1
80 FOR X=2 TO 14
90 FOR Y=2 TO 14
110 IF RND(0)>.4 THEN A(X,Y)=1
120 B(X,Y)=A(X,Y)
130 NEXT Y
140 NEXT X
150 GOSUB 330
160 G=G+1
170 FOR X=2 TO 14
180 FOR Y=2 TO 14
190 C=0
200 IF A(X-1,Y-1)=1 THEN C=C+1
210 IF A(X-1,Y)=1 THEN C=C+1
220 IF A(X-1,Y+1)=1 THEN C=C+1
230 IF A(X,Y-1)=1 THEN C=C+1
240 IF A(X,Y+1)=1 THEN C=C+1
250 IF A(X+1,Y-1)=1 THEN C=C+1
260 IF A(X+1,Y)=1 THEN C=C+1
270 IF A(X+1,Y+1)=1 THEN C=C+1
280 IF A(X,Y)=1 AND C<>3 AND C<>2 THEN B(X,Y)=0
290 IF A(X,Y)=0 AND C=3 THEN B(X,Y)=1
300 NEXT Y
```

```
310 NEXT X
320 GOTO 150
330 PRINT @ 9,"GENERATION ";G
340 SOUND (G/2+1),2
350 PRINT:PRINT "    ";
390 FOR X=2 TO 14
400 FOR Y=2 TO 14
410 A(X,Y)=B(X,Y)
420 IF A(X,Y)=1 THEN PRINT "* "; ELSE PRINT "  ";
440 NEXT Y
450 PRINT:PRINT "    ";
460 NEXT X
470 RETURN
```

NIMGRAB

This is a game which seems easy at first, but is the very devil to play and win. The program was prepared for this book by Glen Pringle.

You and the wily computer take it in turns to take objects away from the ones on the screen. There is a limit to how many you can take away each move. This limit does not change within a game, although it changes from round to round. The winner is the person who forces the other player to remove the last object.

Once you see it in action, you'll understand how to play it. Then you can start working on perfecting a method of actually beating the computer.

```
10 REM NIMGRAB
20 CLS
30 M=0:E=0:Z=RND(8)+16
40 IF 2*INT(Z/2)=Z THEN Z=Z+1
50 H=3+RND(2)
60 PRINT "THE MAXIMUM YOU CAN GRAB IS"H
70 GOSUB 270
80 IF E>0 THEN PRINT "YOU TOOK"E", AND I TOOK"Q
90 FOR K=1 TO Z
100 PRINT K;:IF RND(10)>8 THEN PRINT
110 NEXT K
120 GOSUB 270
130 PRINT "AND HOW MANY WILL YOU GRAB?"
135 P$=INKEY$
140 E=VAL(INKEY$)
150 IF E<1 OR E>H OR E>Z THEN 140
160 PRINT:PRINT"SO YOU WANT TO GRAB"E
170 Z=Z-E
```

```
180 GOSUB 270
190 IF Z<1 THEN GOTO 320
200 Q=Z-1-INT((Z-1)/(H+1))*(H+1)-INT(RND(2))+INT(RND(2))
210 IF Q<1 OR Q>H THEN 200
220 GOSUB 270
230 Z=Z-Q
240 IF Z<1 THEN PRINT"I GRABBED THE LAST ONE SO YOU   WIN!!":END
250 GOSUB 270
260 GOTO 60
270 PRINT
280 PRINT "*******************************"
290 PRINT
300 FOR O=1 TO 500:NEXT O
310 RETURN
320 PRINT "YOU GRABBED THE LAST ONE "
330 PRINT ,"SO I WIN!!":FOR Z=1 TO 500:NEXT Z
340 SOUND 16,3:SOUND 16,2:SOUND 13,4:SOUND 18,2:SOUND 16,5
350 SOUND 13,6:END
```

1024×
8
8192

1024×
64
4096
61440
71536

LAS VEGAS HIGH

Here's a slot machine program for you to run. You need only decide how much you'll bet before 'pulling the handle' on the machine, and the reels will whirl away.

Your winnings, as you'd expect, are related to the relative chances of the various combinations coming up. The computer keeps up the chatter as the game, and your wealth (or poverty) unfold.

The program, prepared for this book by Glen Pringle, leads you through the responses required from you.

```
10 REM LAS VEGAS HIGH
20 GOSUB 1110 :REM INITIALISE
30 GOSUB 870 :REM PLAYER INPUT
40 GOSUB 520 :REM OPERATE SLOT MACHINE
50 IF CASH<1 THEN 90
60 IF CASH>2500 THEN 290
70 GOTO 30
80 REM *****
90 REM BROKE
100 REM *****
110 GOSUB 410
120 PRINT "THAT'S THE END OF THE LINE,"
130 PRINT "OH ONCE MIGHTY GAMBLER..."
140 GOSUB 410
150 PRINT "YOU'RE STONE, FLAT BROKE!!"
160 GOSUB 410
170 PRINT "PRESS 'Y' IF YOU'D LIKE TO"
180 PRINT "HAVE ANOTHER GO AT BREAKING"
190 PRINT "!!!!! LAS VEGAS HIGH !!!!!"
200 PRINT "(OR PRESS 'N' IF YOU WISH TO":PRINT ,"LEAVE)"
```

```
210 A$=INKEY$
220 IF A$<>"Y" AND A$<>"N" THEN 210
230 IF A$="Y" THEN RUN
240 PRINT:PRINT "OK, PUNTER..."
250 GOSUB 410
260 PRINT "THANKS FOR THE GAME!"
270 END
280 REM **************
290 REM BROKE THE BANK
300 REM **************
310 GOSUB 410
320 PRINT "WELL DONE, GAMBLER!!"
330 GOSUB 410
340 PRINT "YOU'VE REACHED OUR HOUSE LIMIT"
350 PRINT "SO WE'LL HAVE TO THROW YOU OUT
360 PRINT "PEOPLE WITH LUCK LIKE YOURS GIVE"
380 PRINT "OUR CASINO A BAD NAME......."
390 GOTO 160
400 REM *****
410 REM DELAY
420 REM *****
430 FOR P=1 TO 1000:NEXT P
440 PRINT:PRINT
450 RETURN
460 REM *********
470 REM DELAY TWO
480 REM *********
490 FOR P=1 TO 1000:NEXT P
500 RETURN
510 REM *******************
520 REM OPERATE SLOT MACHINE
530 REM *******************
540 CLS
550 GOSUB 410
560 PRINT "/^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\"
570 PRINT " *";
```

```
580 FOR M=1 TO 3
590 GOSUB 460
600 A=RND(46)
610 IF A<2 THEN PRINT A$(4);:C(M)=1
620 IF A>1 AND A<6 THEN PRINT A$(3);:C(M)=2
630 IF A>5 AND A<12 THEN PRINT A$(1);:C(M)=3
640 IF A>11 AND A<20 THEN PRINT A$(2);:C(M)=4
650 IF A>19 AND A<31 THEN PRINT A$(5);:C(M)=5
660 IF A>30 THEN PRINT A$(6);:C(M)=6
670 PRINT "*";:SOUND 31,1
680 NEXT M
690 GOSUB 410
700 WIN=0
710 IF C(1)+C(2)+C(3)=3 THEN 1220
720 IF C(1)=C(2) AND C(3)=C(2) AND C(2)=2 THEN 1240
730 IF C(1)=C(2) AND C(3)=C(2) AND C(1)<>1 AND C(2)<>3 THEN 1250
740 IF C(1)=C(2) OR C(1)=C(3) OR C(2)=C(3) THEN 1260
750 IF C(3)=2 THENPRINT"A BELL AT THE END IS A BONUS!":WIN=WIN+.6
760 IF C(1)=3 AND C(3)=3 THEN PRINT"AN APPLE AT EACH END IS GOOD"
765 IF C(1)=3 AND C(3)=3 THEN WIN=WIN+.5
770 IF C(1)=4 AND C(2)=3 AND C(1)=4 THEN GOSUB 1270:WIN=WIN+.4
780 GOSUB 410
790 WIN=INT(WIN*BET)
800 IF WIN>0 THEN PRINT "AND YOU'VE WON"WIN"!":CASH=CASH+WIN
810 IF WIN=0 THEN PRINT "AND YOU'VE LOST $"BET:CASH=CASH-BET
820 GOSUB 410
830 IF CASH>0 THEN PRINT "SO NOW YOU HAVE $"CASH
840 GOTO 460
850 REM
860 REM ***********
870 REM PLAYER INPUT
880 REM ***********
890 CLS
900 GOSUB 410
910 IF CASH<300 THEN PRINT "HI THERE, GAMBLER!"
920 IF CASH>299ANDCASH<600 THEN PRINT "YOU'RE DOING WELL TONIGHT"
```

```
930 IF CASH>599 AND CASH<900 THEN PRINT"LADY LUCK HAS CERTAINLY";
935 IF CASH>599 AND CASH<900 THEN PRINT" SMILED ON YOU!"
940 IF CASH>899 AND CASH<1200 THEN PRINT "THE FATES ARE BEING";
945 IF CASH>899 AND CASH<1200 THEN PRINT " EXTREMELY KIND"
950 IFCASH>1199 THENPRINT"IT IS SO GOOD TO SEE AN EXPERT AT WORK"
960 GOSUB 470
970 PRINT:PRINT "YOU HAVE $"CASH
980 PRINT:INPUT "HOW MUCH DO YOU WANT TO BET";BET
990 IF BET>CASH THEN PRINT "YOU AIN'T GOT THAT MUCH!":GOTO 980
1000 GOSUB 410
1010 PRINT "OK, SIR, $"BET"IT IS!"
1020 GOSUB 410
1030 PRINT "PRESS THE SPACE BAR TO PLAY"
1040 IF INKEY$<>" " THEN 1040
1050 FOR T=1 TO 40
1060 PRINT TAB(T/2);"******* STAND BY *******"
1070 PRINT
1080 NEXT T
1090 RETURN
1110 REM INITIALISE
1130 CLS
1140 DIM A$(6),C(6)
1150 CASH=250
1160 FOR B=1 TO 6
1170 READ A$(B)
1180 NEXT B
1190 RETURN
1200 DATA "$APPLE$","#CHERRY#","*BELL*","!!BAR!!"
1210 DATA "<<LEMON>>","[[PLUM]]"
1220 PRINT"THREE BARS!!!":GOSUB410:PRINT"THAT'S JACKPOT STYLE!!"
1230 WIN=WIN+9:GOTO 750
1240 PRINT "THREE BELLS!!!":WIN=WIN+3.9:GOTO 750
1250 PRINT "THREE OF A KIND":WIN=WIN+3.5:GOTO 750
1260 PRINT ">> A PAIR <<":WIN=WIN+.7:GOTO 750
1270 PRINT "THAT OLD 'CHERRY,BELL,CHERRY'"
1280 PRINT "COMBINATION IS ONE OF MY FAVORITES!":RETURN
```

SWITCHEROO

In Switcheroo, you're presented with the digits 1 to 9 arranged in a random order. You have to get them into the 123456789 order in as few moves as possible.

You enter your moves as numbers, and the computer performs a 'switcheroo' using the number you've entered. The program was prepared for this book by Glen Pringle.

```
10 REM SWITCHEROO
20 CLS:B=B+1
30 PRINT:PRINT"PLEASE STAND BY..."
40 GOSUB 100
50 GOSUB 220
60 IF A$="123456789" THEN 390
70 M=M+1
80 GOTO 50
90 END
100 M=1:X=0
110 A$=""
120 FOR T=1 TO 9
130 L=RND(9)+48
140 Q=1
150 IF MID$(A$,Q,1)=CHR$(L) THEN 130
160 IF Q<T THEN Q=Q+1:GOTO 150
170 A$=A$+CHR$(L)
180 PRINT A$
190 PRINT:SOUND 1,1
200 NEXT T
210 RETURN
220 REM PRINT OUT
230 IF M>1 THEN 260 ELSE CLS
240 PRINT:PRINT:PRINT
250 PRINT "MOVE NUMBER"M
```

```
260 PRINT@107,M
270 PRINT@256,A$:SOUND 31,1
280 PRINT:PRINT"WHICH NUMBER TO SWITCHEROO?"
290 IF INKEY$ <>"" THEN 290
300 C$=INKEY$
310 R=VAL(C$)
320 IF R<1 OR R>8 THEN 300
330 B$=""
340 FOR T=9 TO R STEP -1
350 B$=B$+MID$(A$,T,1)
360 NEXT T
370 A$=LEFT$(A$,R-1)+B$
380 RETURN
390 PRINT:PRINT:PRINT
400 PRINT A$
410 PRINT:PRINT:PRINT
420 PRINT"YOU DID IT, CHAMP!"
430 PRINT:PRINT"AND IT TOOK JUST"M"MOVES..."
435 IF B=1 THEN H=M
440 IF M<H THEN H=M
450 PRINT "LOW SCORE-"H
460 FOR I=1 TO 3000:NEXT
470 GOTO 20
```

MANCALA

MANCALA is one of the series of 'pebble-in-pits' games often called names like OWARI, AWARI and KALAH. The game is played from Africa to the Philippines, and now moves into your home, via a very clever computer opponent.

As you can see from the sample game which follows this introduction, the game begins with six 'pits' (the letters A to F and L to G) facing each player. Your pits are those from L to G. Each pit contains three seeds at the beginning of the game. Choosing any pit on your side, you pick up all the seeds from it, and then proceed to move in a clockwise direction, sowing a seed in each pit as you pass it. You do not sow any seeds in the pits at each end of the board, the ones which start off as zeroes.

If your final seed lands opposite an empty pit, then all the seeds in the pit you've landed in become yours, and are transferred to your 'home'. Your home is the zero to the left of the board, the computer's home is the zero to the right.

The game continues until either side is completely empty, so the player cannot move. At this point, the player with the largest number of seeds in his or her home is the winner.

The computer plays well in this game, but with practice you'll learn to defeat it. Just don't expect too many victories in the early stages.

Here's one game against the program:

```
   A  B  C  D  E  F
   3  3  3  3  3  3
0                    0
   3  3  3  3  3  3
   L  K  J  I  H  G
            A  B  C  D  E  F
            3  3  3  3  3  0
         0                    0
            3  3  3  4  4  4
            L  K  J  I  H  G
   A  B  C  D  E  F
   3  3  3  3  3  0
0                    0
   3  4  4  5  5  0
   L  K  J  I  H  G
            A  B  C  D  E  F
            3  3  0  4  4  1
         0                    0
            3  4  4  5  5  0
            L  K  J  I  H  G
   A  B  C  D  E  F
   4  3  0  4  4  1
0                    0
   4  5  5  6  0  0
   L  K  J  I  H  G
            A  B  C  D  E  F
            4  3  0  0  0  2
         0                    5
            4  5  5  6  1  1
            L  K  J  I  H  G
```

This is later in the game:

```
     A   B   C   D   E   F
     7   2   3   3   3   0
 9                           5
     7   0   1   0   2   0
     L   K   J   I   H   G
                 A   B   C   D   E   F
                 7   2   3   0   4   0
             9                           6
                 7   0   1   0   2   1
                 L   K   J   I   H   G
     A   B   C   D   E   F
     7   0   3   0   4   0
11                          6
     7   1   0   0   2   1
     L   K   J   I   H   G
                 A   B   C   D   E   F
                 7   0   0   1   5   1
            11                           7
                 7   1   0   0   2   0
                 L   K   J   I   H   G
     A   B   C   D   E   F
     8   1   1   2   6   0
13                          7
     0   1   0   0   2   1
     L   K   J   I   H   G
                 A   B   C   D   E   F
                 0   2   2   0   7   1
            13                           10
                 0   1   0   1   3   2
                 L   K   J   I   H   G
```

And this is how it ended:

```
     A   B   C   D   E   F
     3   0   0   0   0   0
19                           19
     0   0   1   0   0   0
     L   K   J   I   H   G
                 A   B   C   D   E   F
                 0   1   1   1   0   0
            19                           19
                 0   0   1   0   0   0
                 L   K   J   I   H   G
     A   B   C   D   E   F
     0   0   1   1   0   0
20                           19
     0   1   0   0   0   0
     L   K   J   I   H   G
                 A   B   C   D   E   F
                 0   0   1   0   1   0
            20                           19
                 0   1   0   0   0   0
                 L   K   J   I   H   G
     A   B   C   D   E   F
     0   0   1   0   1   0
20                           19
     1   0   0   0   0   0
     L   K   J   I   H   G
                 A   B   C   D   E   F
                 0   0   1   0   0   1
            20                           19
                 1   0   0   0   0   0
                 L   K   J   I   H   G
```

```
THAT'S THE END OF THE GAME


    A   B   C   D   E   F
    1   0   1   0   0   1
 22
                            19
    0   0   0   0   0   0
    L   K   J   I   H   G



 MY SCORE> 19      22 <YOUR SCORE

 YOU'RE THE WINNER!
```

And here's the listing of MANCALA:

```
10 REM MANCALA
15 REM SHOWING STRUCTURED
16 REM PROGRAMMING TECHNIQUES
20 GOSUB 930:REM INITIALISE
30 GOSUB 770:REM PRINT BOARD
40 FOR P=1 TO 500:NEXT P:REM DELAY
50 GOSUB 160:REM COMPUTER MOVE
60 GOSUB 770:REM PRINT BOARD
70 GOSUB 620:REM HUMAN MOVE
80 CW=0:HW=0
90 FOR C=1 TO 12
100 IF C<7 THEN CW=CW+A(C)
110 IF C>6 THEN HW=HW+A(C)
120 NEXT C
130 IF CW=0 OR HW=0 THEN 510
140 GOTO 30
150 REM ***************
160 REM COMPUTER MOVE
170 GM=0:C=0
```

```
180 C=C + 1
190 IF A(C)=0 THEN 180
200 Z=C + A(C)
210 IF Z>12 THEN Z=Z - 12
215 FLAG=0
220 IF Z>6 THEN IF A(Z-6)<>0 AND A(Z)=0 THEN FLAG=1
225 IF FLAG=1 THEN IF A(Z-6)>GM THEN GM=C
227 FLAG=0
230 IF Z<7 THEN IF A(Z+6)<>0 AND A(Z)=0 THEN FLAG=1
235 IF FLAG=1 THEN IF A(Z+6)>GM THEN GM=C
240 IF C<6 THEN 180
250 IF GM=0 THEN 370
260 C=GM
270 PRINT @ 34,"I'LL MOVE FROM ";CHR$(64+C)," "
272 SOUND RND(0)*20+1,3:SOUND RND(0)*20+1,1:SOUND 5,2
275 PRINT @ 34,"                    ","   "
280 FOR Z=C TO C+A(C)
290 IF Z>12 THEN A(Z-12)=A(Z-12)+1
300 IF Z<13 THEN A(Z)=A(Z)+1
310 NEXT Z
320 Z=C+A(C)-1:IF Z>12 THEN Z=Z-12
330 A(C)=0
340 B(2)=B(2)+A(13-Z):A(13-Z)=0
350 RETURN
360 REM*************
370 REM NON-SCORE MOVE
380 W=0
390 W=W + 1
400 C=INT(RND(0)*6)+1
410 IF A(C) <> 0 THEN 440
420 IF W<20 THEN 390
430 GOTO 510
440 PRINT @ 34,"I'LL MOVE FROM ";CHR$(64+C)
442 SOUND 6,1:SOUND 3,1:SOUND RND(0)*20+1,3:SOUND RND(0)*20+1,1
445 PRINT @ 34,"                    ","   "
450 FOR Z= C TO C+A(C)
460 IF Z<13 THEN A(Z)=A(Z)+1
```

```
470 IF Z>12 THEN A(Z-6)=A(Z-6)+1
480 NEXT Z
490 A(C)=0:GOTO 350
500 REM *************
510 REM END OF GAME
520 GOSUB 770
530 PRINT @ 34,"THAT'S THE END OF THE GAME"
540 SOUND RND(0)*20,1:SOUND RND(0)*20,2
550 IF B(1)>B(2) THEN PRINT @ 480,"YOU'RE THE WINNER!"
560 IF B(1)<B(2) THEN PRINT @ 480,"I'M THE WINNER!"
570 IF B(1)=B(2) THEN PRINT @ 480,"IT LOOKS LIKE A DRAW!"
580 PRINT @ 416,"MY SCORE>";B(2);"  ";B(1);"<YOUR SCORE"
600 GOTO 600
610 REM*****************
620 REM HUMAN MOVE
630 PRINT @ 34,"WHICH PIT TO START WITH?":PRINT @ 66," "
635 INPUT A$:PRINT @ 34,"
                                                      "," "
640 PRINT @ 96,"    ":B=ASC(A$)-64
650 IF B<7 OR B>12 THEN 630
660 CO=B:Z=B+A(B):IF Z>12 THEN Z=Z-12
670 M=A(Z)
680 FOR Z=B TO B+A(B)
690 IF Z>12 THEN A(Z-12)=A(Z-12)+1
700 IF Z<13 THEN A(Z)=A(Z)+1
710 NEXT Z
720 Z=B+A(B)-1:IF Z>12 THEN Z=Z-12
730 IF M=0 THEN B(1)=B(1)+A(13-Z):A(13-Z)=0
740 A(CO)=0
750 RETURN
760 REM****************
770 REM PRINT BOARD
780 PRINT @ 128,"   A  B  C  D  E  F":PRINT "   ";
800 FOR C=1 TO 6
810 PRINT A(C);
820 NEXT C
830 PRINT:PRINT B(1);"
                         "B(2):PRINT "   ";
```

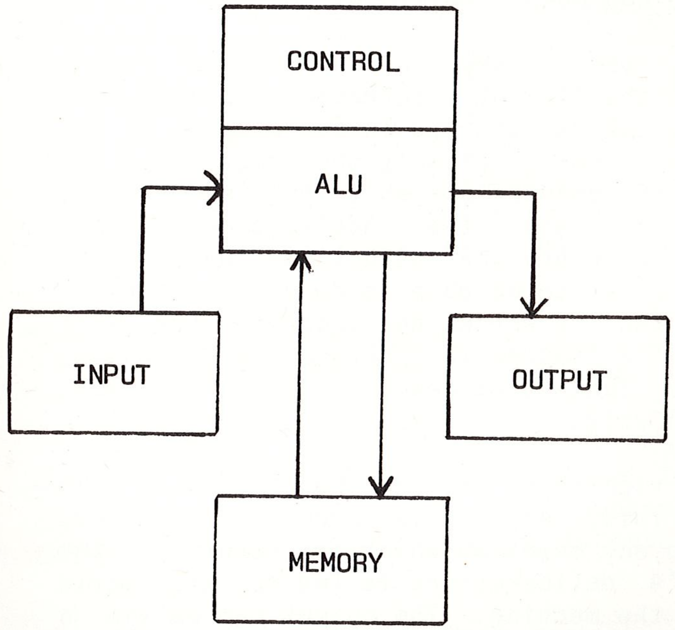
```
840 FOR C=12 TO 7 STEP -1
850 PRINT A(C);
860 NEXT C
870 PRINT:PRINT "    L  K  J  I  H  G"
880 COPY:REM DELETE THIS LINE IF YOU DON'T WANT PRINTER COPY
890 RETURN
900 PRINT @ 34,"I MOVE FROM ";CHR$(64+GM)
910 C=GM
920 REM****************
930 REM INITIALISE
940 CLS
960 DIM A(12),B(12)
970 FOR C=1 TO 12
980 A(C)=3
990 NEXT C
1000 RETURN
```

If you look back to the first few lines of this program, you'll see that it uses structured programming techniques as outlined on pages one to four, in the TIC TAC TOE program.

APPENDICES

APPENDIX ONE - WHAT IS A COMPUTER

You don't need to know what a computer is, or how it works, to make use of it and get pleasure out of using it. In much the same way, you don't need to able to repair a car in order to drive it. However, having some knowledge of the parts of the machine you're using - whether it's a car, a video recorder or a computer - is likely to add to your pleasure in using it.



This diagram shows the five basic parts of a computer. We'll start by looking at the left-hand box, the INPUT. This refers to any means which exist for getting information into the computer from the outside world. It can be a keyboard, a microphone (if the computer is equipped to understand speech), or the computer can read the contents of a magnetic disk, magnetic tape (like the cassettes your computer uses) or from punched tape.

The next box across is the heart of the system, the Arithmetic Logic Unit, or ALU for short. This carries out the mathematical work required, and makes decisions.

Above it you can see the CONTROL UNIT which controls the flow of information through the computer. Below the ALU in our diagram is the memory which holds not only the program and the results of computations while the program is running, but also the 'intelligence' of the machine. Here are the intructions the computer needs in order to be able to do such things as control the TV screen and interpret the word SOUND in a program in order to make a noise. We'll be looking at memory in a little more detail shortly.

On the right-hand side of the diagram is the OUTPUT. This, as you've probably realised, refers to any means by which the result of the computer's deliberations is fed to the world outside the machine. The output can be via a number of things such as a TV screen, a printer or a speech unit.

In many computers today, the CONTROL UNIT and the ALU come together on a single chip, called

a CENTRAL PROCESSING UNIT, or CPU. This means that only input, output and a power supply have to be added to turn the CPU into a computer.

So, in its most primitive form, the computer is a device which operates upon numbers under the control of other numbers. The computer has a central processing unit which actually carries out the numerical operations, a memory where a mixture of instructions and data is stored, and some way of getting information in from the outside world, and of sending results out.

Computers contain two sorts of memory. The first kind is fixed when you buy the computer. This sort of memory is called 'Read Only Memory' because all the computer can do is read it and act in accordance with the instructions written in this memory area. It is called ROM for short.

As well as ROM, your computer (in common with all other computers) has memory which can be modified when the computer is running. This memory - called 'Random Access Memory' (or RAM) - is the place where the program you type in, or load in from a tape, resides. RAM contents are lost (the computer, in effect, 'forgets') when the power is turned off.

So, ROM is fixed memory, which tells the computer how to do what it needs to do when running (such as how to read your depressions on the keyboard, and how to add numbers together), and RAM is changeable memory which holds the program you are currently running and the intermediate results of that program.

When you program your computer, you do so in the computer language known as BASIC. The advantage of BASIC to human beings is that it is reasonably like English, so is fairly easy to work with. However, the computer does not understand BASIC. It only understands, and thinks in, numbers. So, within the ROM is the means by which the computer changes a BASIC program line, such as A = RND(0)*20, into a sequence of numbers it can understand.

As well as the kind of 'built in' memory we've been talking about, there is external memory, which is often called mass storage, with the peripherals used to store this memory being known as MASS STORAGE DEVICES (or MSD). The cheapest MSD, the one you probably use with your computer, is the domestic cassette recorder. Although tape is cheap, it is slow and 'sequential', that is, you have to search through a tape item by item in order to find the one you want.

The other most common sort of MSDs are those in the disk family. FLOPPY DISKS are flexible disks, about the size of a 45 rpm record, coated with magnetic material. They are permanently sealed in square envelopes within which they are free to rotate. The disks are used in conjuction with 'floppy disk drives' which contain magnetic heads which can detect and create magnetic patterns on the disks.

HARD DISKS, which are larger and more expensive

than floppy ones, can hold much more information than floppies. A typical floppy can hold between 100K and 800K. Hard disks can hold hundreds of thousands of kilobytes of information.

APPENDIX TWO - GLOSSARY OF TERMS

Accumulator - part of the computer's logic unit which stores the intermediate results of computations

Address - a number which refers to a location, generally in the computer's memory, where information is stored

Algorithm - the sequence of steps used to solve a problem

Alphanumeric - generally used to describe a keyboard, and signifying that the keyboard has alphabetical and numerical keys. A numeric keypad, by contrast, only has keys for the digits one to nine, with some additional keys for arithmetic operations, much like a calculator

APL - this stands for Automatic Programming Language, a language developed by Iverson in the early 1960s, which supports a large set of operators and data structures. It uses a non-standard set of characters

Application software - these are programs which are tailored for a specific task, such as word processing, or to handle mailing lists

ASCII - stands for American Standard Code for Information Exchange. This is an almost universal code for letters, numbers and symbols, which has a number between 0 and 255 assigned to each of these, such as 65 for the letter A

Assembler - this is a program which converts another program written in an assembly language (which is a computer program in which a single instruction, such as ADD, converts into a single instruction for the computer) into the language the computer uses directly

BASIC - stands for Beginner's All-purpose Symbolic Instruction Code, the most common language used on microcomputers. It is easy to learn, with many of its statements being very close to English

Baud - a measure of the speed of transfer of data. It generally stands for the number of bits (discrete units of information) per second

Benchmark - a test which is used to measure some aspect ofthe performance of a computer, which can be compared to the result of running a similar test on a different computer

Binary - a system of counting in which there are only two symbols, 0 and 1 (as opposed to the ordinary decimal system, in which there are ten symbols, 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9). Your computer 'thinks' in binary

Bit - an abbreviation of 'binary digit', the smallest discrete unit the computer can understand, and having a value of 0 or 1

Boolean Algebra - the algebra of decision-making and logic, developed by English mathematician George Boole, and at the heart of your computer's ability to make decisions

Bootstrap - a program, run into the computer when it is first turned on, which puts the computer into the state where it can accept and understand other programs

Buffer - a storage mechanism which holds input from a device such as keyboard, then releases it at a rate which the computer dictates

Bug - an error in a program

Bus - a group of electrical connections used to link a computer with an ancillary device, or another computer

Byte - the smallest group of bits (see bit) which makes up a computer word. Generally a computer is descibed as being 'eight bit' or '16 bit', meaning the word consists of a combination of eight or sixteen zeroes or ones

Central Processing Unit (CPU) - the heart of the computer, where arithmetic, logic and control functions are carried out

Character code - the number in ASCII (see ASCII) which refers to a particular symbol, such as 32 for a space and 65 for the letter 'A'

COBOL - stands for Common Business Orientated Language, a standard programming language, close to English, which is used primarily for business

Compiler - a program which translates a program written in a high level (human-like) language into a machine language which the computer is able to understand directly

Concatenate - to add (adding two strings together is known as 'concatenation')

CP/M - these initials stand for Control Program/Microcomputer, an almost universal disk operating system developed and marketing by Digital Research, Pacific Grove, California

Data - a general term for information processed by a computer

Database - a collection of data, organised to permit rapid access by computer

Debug - to remove bugs (errors) from a program

Disk - a magnetic storage medium (further described as a 'hard disk', 'floppy disk' or even 'floppy') used to store computer information and programs. The disks resemble, to a limited extent, 45 rpm sound records, and are generally eight, five and a quarter, or three and a half inches in diameter. Smaller 'microdisks' are also available for some systems

Documentation - the written instructions and explanations which accompany a program

DOS - stands for Disk Operating System (and generally pronounced 'doss'), the versatile program which allows a computer to control a disk system

Dot-matrix printer - a printer which forms the letters and symbols by a collection of dots, usually on an eight by eight, or seven by five, grid

Double-density - adjective used to describe disks when recorded using a special technique which, as the name suggests, doubles the amount of storage the disk can provide

Dynamic memory - computer memory which requires constant recharging to retain its contents

EPROM - stands for Erasable Programmable Read Only Memory, a device which contains computer information in a semi-permanent form, demanding sustained exposure to ultra-violet light to erase its contents

Error messages - information from the computer to the user, sometimes consisting only of numbers or a few letters, but generally of a phrase (such as 'Out of memory') which points out a programming or operational error which has caused the computer to halt program execution

Field - A collection of characters which form a distinct group, such as an indentifying code, a name or a date; a field is generally part of a record

File - A group of related records which are processed together, such as an inventory file or a student file

Firmware - The solid components of a computer system are often called the 'hardware', the programs, in machine-readable form on disk or cassette, are called the 'software', and programs which are hardwired into a circuit, are called 'firmware'. Firmware can be altered, to a limited extent, by software in some circumstances

Flag - this is an indicator within a program, with the 'state of the flag' (i.e. the value it holds) giving information regarding a particular condition

Floppy disk - see disk

Flowchart - this is a written layout of program structure and flow, using various shapes, such as a rectangle with sloping sides for a computer action, and a diamond for a computer decision. A flowchart is generally written before any lines of program are entered into the computer

FORTRAN - a high level computer language, generally used for scientific work (from FORmula TRANslation)

Gate - a computer 'component' which makes decisions, allowing the circuit to flow in one direction or another, depending on the conditions to be satisfied

GIGO - acronym for 'Garbage In Garbage Out', suggesting that if rubbish or wrong data is fed into a computer, the result of its processing of such data (the output) must also be rubbish

Global - a set of conditions which effects the entire program is called 'global', as opposed to 'local'

Graphics - a term for any output of computer which is not alphanumeric, or symbolic

Hard copy - information dumped to paper by a printer

Hardware - the solid parts of the computer (see 'software' and 'firmware')

Hexadecimal - a counting system often used by machine code programmers because it is closely related to the number storage methods used by computers, based on the number 16 as opposed to our 'ordinary' number system which is based on 10)

Hex pad - a keyboard, somewhat like a calculator, which is used for direct entry of hexadecimal numbers

High-level langauges - programming languages which are close to English. Low-level languages are closer to those which the computer understands. Because high-level languages have to be compiled into a form which the computer can understand before they are processed, high-level languages run more slowly than do their low-level counterparts

Input - any information which is fed into a program during execution

I/O - stands for Input/Output port, a device the computer uses to communicate with the outside world

Instruction - an element of programming code, which tells the computer to carry out a specific task. An instruction in assembler language, for example, is ADD which (as you've probably guessed) tells the computer to carry out an addition

Interpreter - converts the high-level ('human-understandable') program into a form which the computer can understand

Joystick - an analogue device which feeds signal into a computer which is related to the position which the joystick is occupying; generally used in games programs

Kilobyte - the unit of language measurement; one kilobyte (generally abbreviated as K) equals 1024 bytes

Line printer - a printer which prints a complete line of characters at one time

Low-level language - a language which is close to that used within the computer (see high-level language)

Machine language - the step below a low-level language; the language which the computer understands directly

Memory - the device or devices used by a computer to hold information and programs being

1, 2, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768

currently processed, and for the instruction set fixed within a computer which tells it how to carry out the demands of the program. There are basically two types of memory (see RAM and ROM)

Microprocessor - the 'chip' which lies at the heart of your computer. This does the 'thinking'

Modem - stands for MOdulator/DEModulator, and is a device which allows one computer to communicate with another via the telephone

Monitor - (a) a dedicated television-screen for use as a computer display unit, contains no tuning apparatus; (b) the information within a computer which enables it to understand and execute program instructions

Motherboard - a unit, generally external, which has slots to allow additional 'boards' (circuits) to be plugged into the computer to provide facilities (such as high-resolution graphics, or 'robot control') which are not provided with the standard machine

Mouse - a control unit, slightly smaller than a box of cigarettes, which is rolled over the desk, moving an on-screen cursor in parallel to select options and make decisions within a program. 'Mouses' work either by sensing the action of their wheels, or by reading a grid pattern on the surface upon which they are moved

Network - a group of computers working in tandem

Numeric pad - a device primarily for entering numeric information into a computer, similar to a calculator

Octal - a numbering system based on eight (using the digits 0, 1, 2, 3, 4, 5, 6 and 7)

On-line - device which is under the direct control of the computer

Operating system - this is the 'big boss' program or series of programs within the computer which controls the computer's operation, doing such things as calling up routines when they are needed and assigning priorities

Output - any data produced by the computer while it is processing, whether this data is displayed on the screen or dumped to the printer, or is used internally

Pascal - a high level language, developed in the late 1960s by Niklaus Wirth, which encourages disciplined, structured programming

Port - an output or input 'hole' in the computer, through which data is transferred

Program - the series of instructions which the computer follows to carry out a predetermined task

PILOT - a high level language, generally used to develop computer programs for education

RAM - stands for Random Access Memory, and is

the memory on board the computer which holds the current program. The contents of RAM can be changed, while the contents of ROM (Read Only Memory) cannot be changed under software control

Real-time - when a computer event is progressing in line with time in the 'real world', the event is said to be occurring in real time. An example would be a program which showed the development of a colony of bacteria which developed at the same rate that such a real colony would develop. Many games, which require reactions in real time, have been developed. Most 'arcade action' programs occur in real time

Refresh - The contents of dynamic memories (see memory) must receive periodic bursts of power in order for them to maintain their contents. The signal which 'reminds' the memory of its contents is called the refresh signal

Register - a location in computer memory which holds data

Reset - a signal which returns the computer to the point it was in when first turned on

ROM - see RAM

RS-232 - a standard serial interface (defined by the Electronic Industries Association) which connects a modem and associated terminal equipment to a computer

S-100 bus - this is also a standard interface (see RS-232) made up of 100 parallel common communication lines which are used to connect circuit boards within micro-computers

SNOBOL - a high level language, developed by Bell Laboratories, which uses pattern recognition and string manipulation

Software - the program which the computer follows (see firmware)

Stack - the end point of a series of events which are accessed on a last in, first out basis

Subroutine - a block of code, or program, which is called up a number of times within another program

Syntax - as in human languages, the syntax is the structure rules which govern the use of a computer language

Systems software - sections of code which carry out administrative tasks, or assist with the writing of other programs, but which are not actually used to carry out the computer's final task

Thermal printer - a device which prints the output from the computer on heat-sensitive paper

Time-sharing - this term is used to refer to a large number of users, on independent terminals, making use of a single computer,

which divides its time between the users in such a way that each of them appears to have the 'full attention' of the computer

Turnkey system - a computer system (generally for business use) which is ready to run when delivered, needing only the 'turn of a key' to get it working

Volatile memory - a memory device which loses its contents when the power supply is cut off

Word processor - a dedicated computer (or a computer operating a word processing program) which gives access to an 'intelligent typewriter' with a large range of correction and adjustment features

Now that you've mastered simple BASIC programming on your Dick Smith VZ200, what do you do next?

The answer lies in this book. Tim Hartnell, the co-ordinator of the VZ200 Users' Club, and co-author of the book GETTING STARTED WITH YOUR VZ200, turns his attention to a number of fascinating areas of computer programming, including:

THE SECRET OF ANIMATION AND MOVING GRAPHICS

HOW TO USE MODE 1

MAKING MUSIC

PEEK AND POKE

As well as the 'lessons', there are fifteen or so great games programs you're sure to enjoy running, including ROMTHELLO, V-WING SPACE BATTLE, CHECKERS, LAS VEGAS HIGH and the prehistoric game, MESOZOIC ATTACK.

This book is the key to moving beyond simple BASIC programming with your Dick Smith VZ200.